4th International Conference on Formal Structures for Computation and Deduction

FSCD 2019, June 24–30, 2019, Dortmund, Germany

Edited by

Herman Geuvers

Editors

Herman Geuvers
Radboud University Nijmegen, The Netherlands
Technical University Eindhoven, The Netherlands
herman@cs.ru.nl

ACM Classification 2012
Theory of computation → Models of computation; Theory of computation → Formal languages and automata theory; Theory of computation → Logic; Theory of computation → Semantics and reasoning; Software and its engineering → Language features; Software and its engineering → Formal language definitions; Software and its engineering → Formal methods

ISBN 978-3-95977-107-8

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-107-8.

Publication date
June, 2019

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

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

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

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

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/LIPIcs.FSCD.2019.0

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.

Editorial Board

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

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

Preface
Herman Geuvers 0:ix–0:x

Steering Committee
.... 0:xi

Program Committee
.... 0:xiii

External Reviewers
.... 0:xv

List of Authors
.... 0:xvii–0:xix

Invited Talk

A Fresh Look at the λ-Calculus
Beniamino Accattoli 1:1–1:20

A Linear Logical Framework in Hybrid
Amy P. Felty 2:1–2:2

Extending Maximal Completion
Sarah Winkler 3:1–3:15

Some Semantic Issues in Probabilistic Programming Languages
Hongseok Yang 4:1–4:6

Regular Paper

Bicategories in Univalent Foundations
Benedikt Ahrens, Dan Frumin, Marco Maggesi, and Niels van der Weide 5:1–5:17

Modular Specification of Monads Through Higher-Order Presentations
Benedikt Ahrens, André Hirschowitz, Ambroise Lafont, and Marco Maggesi 6:1–6:19

Towards the Average-Case Analysis of Substitution Resolution in λ-Calculus
Maciej Bendkowski 7:1–7:21

Deriving an Abstract Machine for
Strong Call by Need
Małgorzata Biernacka and Witold Charatonik 8:1–8:20

Dependency Pairs Termination in Dependent Type Theory Modulo Rewriting
Frédéric Blanqui, Guillaume Genestier, and Olivier Hermant 9:1–9:21

A Generic Framework for Higher-Order Generalizations
David M. Cerna and Temur Kutsia 10:1–10:19

Homotopy Canonicity for Cubical Type Theory
Thierry Coquand, Simon Huber, and Christian Sattler 11:1–11:23

4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019).
Editor: Herman Geuvers
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Polymorphic Higher-Order Termination
Łukasz Czajka and Cynthia Kop 12:1–12:18

On the Taylor Expansion of Probabilistic λ-terms
Ugo Dal Lago and Thomas Leventis 13:1–13:16

Proof Normalisation in a Logic Identifying Isomorphic Propositions
Alejandro Díaz-Caro and Gilles Dowek 14:1–14:23

λ!-calculus, Intersection Types, and Involutions
Alberto Ciaffaglione, Pietro Di Gianantonio, Furio Honsell, Marina Lenisa, and Ivan Scagnetto 15:1–15:16

Template Games, Simple Games, and Day Convolution
Clovis Eberhart, Tom Hirschowitz, and Alexis Laouar 16:1–16:19

Differentials and Distances in Probabilistic Coherence Spaces
Thomas Ehrhard 17:1–17:17

Modal Embeddings and Calling Paradigms
José Espírito Santo, Luís Pinto, and Tarmo Uustalu 18:1–18:20

Probabilistic Rewriting: Normalization, Termination, and Unique Normal Forms
Claudia Faggian 19:1–19:25

A Linear-Logical Reconstruction of Intuitionistic Modal Logic S4
Yosuke Fukuda and Akira Yoshimizu 20:1–20:24

Sparse Tiling Through Overlap Closures for Termination of String Rewriting
Alfons Geser, Dieter Hofbauer, and Johannes Waldmann 21:1–21:21

Proof Nets for First-Order Additive Linear Logic
Willem B. Heijltjes, Dominic J. D. Hughes, and Lutz Straßburger 22:1–22:22

The Sub-Additives: A Proof Theory for Probabilistic Choice extending Linear Logic
Ross Horne 23:1–23:16

A Lower Bound of the Number of Rewrite Rules Obtained by Homological Methods
Mirai Ikebuchi 24:1–24:18

Gluing for Type Theory
Ambrus Kaposi, Simon Huber, and Christian Sattler 25:1–25:20

The Discriminating Power of the Let-In Operator in the Lazy Call-by-Name Probabilistic λ-Calculus
Simona Kašterović and Michele Pagani 26:1–26:20

Hilbert's Tenth Problem in Coq
Dominique Larchey-Wendling and Yannick Forster 27:1–27:20

The Δ-calculus: Syntax and Types
Luigi Liquori and Claude Stolze 28:1–28:20

Pointers in Recursion: Exploring the Tropics
Paulin Jacobé de Naurois 29:1–29:18

Typed Equivalence of Effect Handlers and Delimited Control
Maciej Piróg, Piotr Polesiuk, and Filip Sieczkowski 30:1–30:16

Cubical Syntax for Reflection-Free Extensional Equality
Jonathan Sterling, Carlo Angiuli, and Daniel Gratzer 31:1–31:25

Guarded Recursion in Agda via Sized Types
Niccolò Veltri and Niels van der Weide 32:1–32:19

Sequence Types for Hereditary Permutators
Pierre Vial 33:1–33:15

System Description

Model Checking Strategy-Controlled Rewriting Systems
Rubén Rubio, Narciso Martí-Oliet, Isabel Pita, and Alberto Verdejo 34:1–34:18

Preface

The 4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019) was held June 24–30, 2019 in Dortmund, Germany. FSCD (http://fscd-conference.org/) covers all aspects of formal structures for computation and deduction, from theoretical foundations to applications. Building on two communities, RTA (Rewriting Techniques and Applications) and TLCA (Typed Lambda Calculi and Applications), FSCD embraces their core topics and broadens their scope to closely related areas in logics and proof theory, new emerging models of computation (e.g. homotopy type theory or quantum computing), semantics and verification in new challenging areas (e.g. blockchain protocols or deep learning algorithms). The FSCD program featured four invited talks given by Beniamino Accattoli (Inria, Paris, France), Amy Felty (University of Ottawa, Canada), Sarah Winkler (University of Innsbruck, Austria) and Hongseok Yang (KAIST, South Korea). FSCD 2019 received 69 submissions with contributing authors from 18 countries. The program committee consisted of 32 members from 18 countries. Each submitted paper has been reviewed by at least three PC members with the help of 105 external reviewers. The reviewing process, which included a rebuttal phase, took place over a period of eight weeks. A total of 30 papers, 29 regular research papers and one system description paper, were accepted for publication and are included in these proceedings.

The Program Committee awarded two FSCD 2019 Best Paper Awards for Junior Researchers: Mirai Ikebuchi for the paper “A Lower Bound of The Number of Rewrite Rules Obtained by Homological Methods”, and Jonathan Sterling, Carlo Angiuli and Daniel Gratzer for the paper “Cubical Syntax for Reflection-Free Extensional Equality”. Mirai Ikebuchi is a graduate student and Jonathan Sterling, Carlo Angiuli and Daniel Gratzer are all PhD students.

In addition to the main program, 7 FSCD-associated workshops were planned on days before and mostly after the conference:

- 8th International Workshop on Confluence, IWC 2019,
- 10th Workshop on Higher Order Rewriting, HOR 2019,
- IFIP Working Group 1.6: Rewriting IFIP Meeting 2019 - 22nd edition,
- 3d Workshop on Trends in Linear Logic and Applications, TLLA 2019,
- 6th International Workshop on Rewriting Techniques for Program Transformations and Evaluation, WPTE 2019,
- 33nd International Workshop on Unification, UNIF 2019,
- 5th International Workshop on Structures and Deduction, SD 2019.

This volume of the proceedings of FSCD 2019 is published in the LIPIcs series under a Creative Common license: online access is free to all papers and authors retain rights over their contributions. We thank the Leibniz Center for Informatics at Schloss Dagstuhl, in particular Michael Wagner for his efficient and reactive support during the production of these proceedings.

Many people have helped to make FSCD 2019 a successful meeting. On behalf of the Program Committee, I thank the many authors of submitted papers for considering FSCD as a venue for their work and the invited speakers who have agreed to speak at this meeting. The Program Committee and the external reviewers deserve big thanks for their careful review and evaluation of the submitted papers. (The members of the Program Committee and the list of external reviewers can be found on the following pages.) The EasyChair conference management system has been a useful tool in all phases of the work of the Program

Committee. The associated workshops have made a big contribution to the lively scientific atmosphere of this meeting and I thank the workshop organizers for their efforts in bringing their meetings to Dortmund and the local Workshop Chair, Boris Düdder, for making the workshops run smoothly. Jakob Rehof, the Conference Chair for FSCD 2019, deserves a very warm thanks for the over all organisation of FSCD 2019, for the smooth functioning of the meeting and for producing the web site. Sandra Alves, as Publicity Chair, made a great contribution in advertising the Conference. The steering committee, lead by Delia Kesner, provided valuable guidance in setting up this meeting and in ensuring that FSCD will have a bright and enduring future. Finally, I thank all participants of the conference for creating a lively and interesting event.

On behalf of the FSCD community, I would like to thank the City of Dortmund, the TU Dortmund University, and the U-Tower for providing the venue of FSCD 2019. I furthermore would like to thank MAXIMAGO, the Friends of the TU Dortmund, and the Alumni of the Faculty of Informatics (aido) for supporting FSCD 2019. The Center of Excellence Logistics & IT (Leistungszentrum Logistik und IT), is gratefully acknowledged as scientific partner of FSCD 2019.

Herman Geuvers
Program Chair of FSCD 2019

Steering Committee

S. Alves	Porto U.
M. Ayala-Rincón	Brasilia U.
C. Fuhs	Birkbeck, London U.
D. Kesner (Chair)	Paris U.
H. Kirchner	Inria
N. Kobayashi	U. Tokyo
C. Kop	Radboud U.
D. Miller	Inria
L. Ong	Oxford U.
B. Pientka	McGill U.
S. Staton	Oxford U.
J. Vicary	Oxford U.

Program Committee

Zena Ariola	U. of Oregon	USA
Mauricio Ayala Rincón	U. of Brasília	Brazil
Andrej Bauer	U. of Ljubljana	Slovenia
Filippo Bonchi	U. of Pisa	Italy
Sabine Broda	U. of Porto	Portugal
Ugo Dal Lago	U. of Bologna & Inria	Italy
Ugo De'Liguoro	U. of Torino	Italy
Deepak Kapur	U. of New Mexico	USA
Peter Dybjer	Chalmers U. of Technology	Sweden
Maribel Fernández	King's College London	UK
Herman Geuvers (Program Chair)	Radboud U. Nijmegen & TU Eindhoven	Netherlands
Jürgen Giesl	RWTH Aachen	Germany
Nao Hirokawa	JAIST	Japan
Salvador Lucas	U. Politècnica de València	Spain
Aart Middeldorp	U. of Innsbruck	Austria
Frank Pfenning	Carnegie Mellon U.	USA
Brigitte Pientka	McGill U.	Canada
Jaco van de Pol	Aarhus U. & U. Twente	Denmark
Femke van Raamsdonk	VU Amsterdam	Netherlands
Carsten Schürmann	ITU Copenhagen	Denmark
Paula Severi	U. of Leicester	UK
Alexandra Silva	U. College London	UK
Sam Staton	Oxford U.	UK
Thomas Streicher	TU Darmstadt	Germany
Aaron Stump	U. of Iowa	USA
Nicolas Tabareau	Inria	France
Sophie Tison	U. of Lille	France
Alwen Tiu	Australian National U.	Australia
Takeshi Tsukada	U. of Tokyo	Japan
Josef Urban	CTU Prague	Czech Republic
Paweł Urzyczyn	U. of Warsaw	Poland
Johannes Waldmann	Leipzig U. of Applied Sciences	Germany

4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019).
Editor: Herman Geuvers
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

External Reviewers

Andreas Abel
Beniamino Accattoli
Ki Yung Ahn
Ariane A. Almeida
María Alpuente
Takahito Aoto
Steffen van Bakel
Paolo Baldan
Franco Barbanera
João Barbosa
Henning Basold
Stefano Berardi
Marc Bezem
Andreas Billig
Frédéric Blanqui
Rafaël Bocquet
Guillaume Bonfante
Chad Brown
Guillaume Brunerie
Paul Brunet
Roberto Bruni
Andrea Corradini
Roy Crole
Łukasz Czajka
Fredrik Dahlqvist
Flavio L. C. De Moura
Mariangiola Dezani
Paul Downen
Burak Ekici
Jacopo Emmenegger
Santiago Escobar
Claudia Faggian
Bertram Felgenhauer
Eric Finster
Marcelo Fiore
Mário Florido
Yannick Forster
Florian Frohn
Thibault Gauthier
Silvia Ghilezan
Raúl Gutiérrez
Masahiro Hamano
Marcel Hark
Willem Heijltjes
Jera Hensel
Dieter Hofbauer
Simon Huber
Jelena Ivetić
Ambrus Kaposi
G. A. Kavvos
Edon Kelmendi
Delia Kesner
Helene Kirchner
Jetty Kleijn
Yuji Kobayashi
Cynthia Kop
James Laird
Isabella Larcher
Andreas Lochbihler
Tim Lyon
Philippe Malbos
Giulio Manzonetto
Narciso Martí-Oliet
Paul-André Melliès
Étienne Miquey
Rasmus Ejlers Møgelberg
Gopalan Nadathur
Keiko Nakata
Daniele Nantes-Sobrinho
Renato Neves
Andreas Nuyts
Pascal Ochem
Vincent van Oostrom
Dominic Orchard
David Pereira
Paolo Pistone
Thiago M. F. Ramos
Yves Roos
Matteo Sammartino
Davide Sangiorgi
Ulrich Schöpp
Sibylle Schwarz
Jens Seeber
Michael Shulman
Kiraku Shintani
Sergey Slavnov
Jonathan Sterling
Claude Stolze
René Thiemann
Riccardo Treglia
Tarmo Uustalu
Gabriele Vanoni
Pedro Vasconcelos
Niccolò Veltri
Daniel Ventura
Andrea Vezzosi
Alicia Villanueva
Niels Voorneveld
Niels van der Weide
Jonathan Weinberger
Freek Wiedijk
Sarah Winkler
Akihisa Yamada
Hans Zantema
Steve Zdancewic

4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019).
Editor: Herman Geuvers
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

List of Authors

Beniamino Accattoli (1)
Inria & LIX, École Polytechnique, UMR 7161, France

Benedikt Ahrens (5, 6)
School of Computer Science, University of Birmingham, United Kingdom

Carlo Angiuli (31)
Carnegie Mellon University, Pittsburgh, USA

Maciej Bendkowski (7)
Jagiellonian University, Faculty of Mathematics and Computer Science, Theoretical Computer Science Department, ul. Prof. Łojasiewicza 6, 30 - 348 Kraków, Poland

Małgorzata Biernacka (8)
Institute of Computer Science, University of Wrocław, Poland

Frédéric Blanqui (9)
INRIA, France; LSV, ENS Paris-Saclay, CNRS, Université Paris-Saclay, France

David M. Cerna (10)
FMV and RISC, Johannes Kepler University Linz, Austria

Witold Charatonik (8)
Institute of Computer Science, University of Wrocław, Poland

Alberto Ciaffaglione (15)
Department of Mathematics, Computer Science and Physics, University of Udine, Italy

Thierry Coquand (11)
Department of Computer Science and Engineering, University of Gothenburg, Sweden

Łukasz Czajka (12)
Faculty of Informatics, TU Dortmund, Germany

Ugo Dal Lago (13)
University of Bologna, Italy; INRIA Sophia Antipolis, France

Pietro Di Gianantonio (15)
Department of Mathematics, Computer Science and Physics, University of Udine, Italy

Gilles Dowek (14)
Inria, LSV, ENS Paris-Saclay, France

Alejandro Díaz-Caro (14)
Instituto de Ciencias de la Computación (CONICET-Universidad de Buenos Aires), Ciudad Autónoma de Buenos Aires, Argentina; Universidad Nacional de Quilmes, Bernal (Buenos Aires), Argentina

Clovis Eberhart (16)
National Institute of Informatics, Tokyo, Japan

Thomas Ehrhard (17)
CNRS, IRIF, Université de Paris, France

José Espírito Santo (18)
Centre of Mathematics, University of Minho, Portugal

Claudia Faggian (19)
Université de Paris, IRIF, CNRS, F-75013 Paris, France

Amy P. Felty (2)
University of Ottawa, Canada

Yannick Forster (27)
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany

Dan Frumin (5)
Institute for Computation and Information Sciences, Radboud University, Nijmegen, The Netherlands

Yosuke Fukuda (20)
Graduate School of Informatics, Kyoto University, Japan

Guillaume Genestier (9)
LSV, ENS Paris-Saclay, CNRS, Université Paris-Saclay, France; MINES ParisTech, PSL University, Paris, France

Alfons Geser (21)
HTWK Leipzig, Germany

Daniel Gratzer (31)
Aarhus University, Denmark

Willem B. Heijltjes (22)
University of Bath, United Kingdom

Olivier Hermant (9)
MINES ParisTech, PSL University, Paris, France

André Hirschowitz (6)
Université Côte d'Azur, CNRS, LJAD, Nice, France

4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019).
Editor: Herman Geuvers
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Tom Hirschowitz (16)
Univ. Grenoble Alpes, Univ. Savoie Mont Blanc, CNRS, LAMA, 73000, Chambéry, France

Dieter Hofbauer (21)
ASW - Berufsakademie Saarland, Germany

Furio Honsell (15)
Department of Mathematics, Computer Science and Physics, University of Udine, Italy

Ross Horne (23)
Computer Science and Communications, University of Luxembourg, Esch-sur-Alzette, Luxembourg

Simon Huber (11, 25)
Department of Computer Science and Engineering, University of Gothenburg, Sweden

Dominic J. D. Hughes (22)
Logic Group, UC Berkeley, USA

Mirai Ikebuchi (24)
Massachusetts Institute of Technology, Computer Science and Artificial Intelligence Laboratory, Cambridge, USA

Paulin Jacobé de Naurois (29)
CNRS, Université Paris 13, Sorbonne Paris Cité, LIPN, UMR 7030, F-93430 Villetaneuse, France

Ambrus Kaposi (25)
Eötvös Loránd University, Budapest, Hungary

Simona Kašterović (26)
Faculty of Technical Sciences, University of Novi Sad , Trg Dositeja Obradovića 6, 21000 Novi Sad, Serbia

Cynthia Kop (12)
Institute of Computer Science, Radboud University Nijmegen, The Netherlands

Temur Kutsia (10)
RISC, Johannes Kepler University Linz, Austria

Ambroise Lafont (6)
IMT Atlantique, Inria, LS2N CNRS, Nantes, France

Alexis Laouar (16)
Univ. Grenoble Alpes, Univ. Savoie Mont Blanc, CNRS, LAMA, 73000, Chambéry, France

Dominique Larchey-Wendling (27)
Université de Lorraine, CNRS, LORIA, Vandœuvre-lès-Nancy, France

Marina Lenisa (15)
Department of Mathematics, Computer Science and Physics, University of Udine, Italy

Thomas Leventis (13)
INRIA Sophia Antipolis, France

Luigi Liquori (28)
Université Côte d'Azur, Inria, Sophia Antipolis, France

Marco Maggesi (5, 6)
Dipartimento di Matematica e Informatica "Dini", Università degli Studi di Firenze, Italy

Narciso Martí-Oliet (34)
Facultad de Informática, Universidad Complutense de Madrid, Spain

Michele Pagani (26)
IRIF, University Paris Diderot - Paris 7, France

Luís Pinto (18)
Centre of Mathematics, University of Minho, Portugal

Maciej Piróg (30)
University of Wrocław, Poland

Isabel Pita (34)
Facultad de Informática, Universidad Complutense de Madrid, Spain

Piotr Polesiuk (30)
University of Wrocław, Poland

Rubén Rubio (34)
Facultad de Informática, Universidad Complutense de Madrid, Spain

Christian Sattler (11, 25)
Department of Computer Science and Engineering, University of Gothenburg, Sweden

Ivan Scagnetto (15)
Department of Mathematics, Computer Science and Physics, University of Udine, Italy

Filip Sieczkowski (30)
University of Wrocław, Poland

Jonathan Sterling (31)
Carnegie Mellon University, Pittsburgh, USA

Claude Stolze (28)
Université Côte d'Azur, Inria, Sophia Antipolis, France

Lutz Straßburger (22)
Inria Saclay, Palaiseau, France; LIX, École Polytechnique, Palaiseau, France

Tarmo Uustalu (18)
School of Computer Science, Reykjavik University, Iceland; Dept. of Software Science, Tallinn University of Technology, Estonia

Niels van der Weide (5, 32)
Institute for Computation and Information Sciences, Radboud University, Nijmegen, The Netherlands

Niccolò Veltri (32)
Department of Computer Science, IT University of Copenhagen, Denmark

Alberto Verdejo (34)
Facultad de Informática, Universidad Complutense de Madrid, Spain

Pierre Vial (33)
Inria, Nantes, France

Johannes Waldmann (21)
HTWK Leipzig, Germany

Sarah Winkler (3)
University of Innsbruck, Austria

Hongseok Yang (4)
School of Computing, KAIST, South Korea

Akira Yoshimizu (20)
French Institute for Research in Computer Science and Automation (INRIA), France

A Fresh Look at the λ-Calculus

Beniamino Accattoli
Inria & LIX, École Polytechnique, UMR 7161, France
beniamino.accattoli@inria.fr

Abstract

The (untyped) λ-calculus is almost 90 years old. And yet – we argue here – its study is far from being over. The paper is a bird's eye view of the questions the author worked on in the last few years: how to measure the complexity of λ-terms, how to decompose their evaluation, how to implement it, and how all this varies according to the evaluation strategy. The paper aims at inducing a new way of looking at an old topic, focussing on high-level issues and perspectives.

2012 ACM Subject Classification Theory of computation → Lambda calculus; Software and its engineering → Functional languages; Theory of computation → Operational semantics

Keywords and phrases λ-calculus, sharing, abstract machines, type systems, rewriting

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.1

Category Invited Talk

Funding *Beniamino Accattoli*: This work has been partially funded by the ANR JCJC grant COCA HOLA (ANR-16-CE40-004-01).

Acknowledgements To Herman Geuvers, Giulio Guerrieri, and Gabriel Scherer for comments on a first draft.

1 Introduction

The λ-calculus is as old as computer science. Many programming languages incorporate its key ideas, many proof assistants are built on it, and various research fields – such as proof theory, category theory, or linguistics – use it as a tool. Books are written about it, it is taught in most curricula on theoretical computer science, and it is the basis of many fashionable trends in programming languages such as probabilistic or quantum programming, or gradual type systems. Is there anything left to say about it? The aim of this informal paper is to provide evidence that yes, the theory of λ-calculus is less understood and stable than it may seem and there still are things left to say about it.

A first point is the solution of the schism between Turing machines and the λ-calculus as models of computation. The schism happened mostly to the detriment of the λ-calculus, that became a *niche* model. Despite belonging to computer science, indeed, the whole theory of the λ-calculus was developed paying no attention to cost issues. Adopting a complexity-aware point of view sheds a new light on old topics and poses new fundamental questions.

A second point is the fact that *the* λ-calculus does not exist. Despite books such as Barendregt's [31] and Krivine's [54], devoted to *the* λ-calculus, it is nowadays clear that, even if one sticks to the untyped λ-calculus with no additional features, there are a number of λ-calcu*li* depending at least on whether evaluation is call-by-name, call-by-value, or call-by-need, and – orthogonally – whether evaluation is strong or weak (that is, whether abstraction bodies are evaluated or not) and, when it is weak, whether terms are closed or can also be open. These choices affect the theory and impact considerably on the design of abstract machines. A comparative study of the various dialects allows to identify principles and differences, and to develop a deeper understanding of higher-order computations.

© Beniamino Accattoli;
licensed under Creative Commons License CC-BY

A third point is about decomposing λ-calculi and the role of sharing. More or less ubiquitously, for practical as well as for theoretical reasons, λ-calculi are presented with `let` expressions, that are nothing else but constructs for first-class sharing, logically corresponding to *the* logical rule behind computation, the *cut* rule. Sharing is also essential for the two previous points, complexity-aware understandings of λ-calculi and defining evaluation strategies such as call-by-need. Perhaps surprisingly, there are simple extensions with sharing that have important properties that "the" λ-calculus lacks. Therefore, there is a third *sharing axis* – beyond the strong/weak axis and the call-by-name/value/need axis – along which the λ-calculus splits into various calculi, that should gain the same relevant status of the other λ-calculi.

This paper tries to explain and motivate these points, through an informal overview of the author's research. The unifying theme is the ambition to harmonise together theoretical topics such as denotational semantics or advanced rewriting properties and practical studies such as abstract machines and call-by-need, using as key ingredients sharing and cost analyses.

2 The Higher-Order Time Problem

The *higher-order time problem* is the problem of how to measure the time complexity of programs expressed as λ-terms. It is a subtle and neglected problem. There are indeed no traces of the question in the two classic books on the λ-calculus, Barendegt's and Krivine's, for instance.

Size Explosion. The λ-calculus has only one computational rule, β-reduction, and the question is whether it is possible to take the number of β-steps as a reasonable time measure or not. Of course, the question is subtler than that, because a λ-term in general admits many different evaluation sequences, of different length, so that one should rather first fix an evaluation strategy. Unfortunately, we have to first deal with a deeper difficulty that affects every strategy.

The essence of the problem indeed amounts to a degeneracy, called *size explosion*: there are families of programs $\{t_n\}_{n\in\mathbb{N}}$ such that t_n evaluates in n β steps to a result s_n of size exponential in n (and in the size $|t_n|$ of t_n). Now, it seems that the number of β-steps cannot be a reasonable measure, because n – the candidate measure of time complexity – does not even account for the time to write down the exponential result s_n.

Size explosion is extremely robust: there is an exploding family $\{t_n\}_{n\in\mathbb{N}}$ (for details see [6]) such that

- t_n is closed;
- all its evaluation sequences to normal form have the same length;
- strong and weak evaluations produce the same result, in the same number of steps;
- lives in the continuation-passing style fragment of the λ-calculus, what is sometimes considered as a sort of simpler kernel;
- can even be typed with simple types.

Moreover, even if one restricts to, say, terms evaluating to booleans, for which normal forms have constant size, size explosion can still happen *along the way*: evaluation can produce a sub-term of exponential size and then erase it, making the number of steps a doubtful notion of cost model even if it is no longer possible to produce an exponential normal form.

Phrased differently, there are no easy tricks: size explosion is here to stay.

Surprisingly, until very recently size explosion was folklore. There are no traces of it in the major literature on the λ-calculus. It was known that duplications in the λ-calculus can have a nasty behaviour, and this is why the λ-calculus is never implemented as it is defined –

all implementations indeed rely on some form of sharing. In 2006, Dal Lago and Martini started the first conscious exploration of the higher-order time problem [39], having instances of size explosion in mind. But it is only in 2014 that Accattoli and Dal Lago named the degeneracy *size explosion* [17], taking it out from the collective unconscious, and starting a systematic exploration.

Reasonable Cost Models. What does it exactly mean for a cost measure to be reasonable? First of all, one has to fix a computational model M, whose role in our case is played by the λ-calculus. As proposed by Slot and van Emde Boas in 1984 [88], a time cost model for M is *reasonable* when there are simulations of M by Turing machines and of Turing machines by M having overhead bounded by a polynomial of the input and of the number of steps (in the source model). For space, similarly, one requires a linear overhead. The basic idea is to preserve the complexity class ¶, that then becomes *robust*, that is, model-independent. Random access machines for instance are a reasonable model (if multiplication is a primitive operation, they need to be endowed with a logarithmic cost model).

The question becomes: are there reasonable cost models for the λ-calculus? At least for time? In principle, everything can be a cost model, but of course one is mainly interested in the natural one, the number of β-steps. The precise question then is: are there reasonable evaluation strategies, that is, strategies whose number of steps is a reasonable time cost model?

Sharing. The good news is that there are strategies for which size explosion is avoidable, or, rather, it can be circumvented, and so, the answer is yes, reasonable strategies do exist. The price to pay is the shift to a λ-calculus enriched with *sharing* of subterms. Roughly, size explosion is based on the blind duplication of useless sub-terms – by keeping these sub-terms shared, the exponential blow up of the size can be avoided and the number of β-steps can be taken as a reasonable measure of time complexity.

Fix a dialect λ_X of the λ-calculus with a deterministic evaluation strategy $\to_X$, and note $\mathtt{nf}_X(t)$ the normal form of t with respect to $\to_X$. The idea is to introduce an intermediate setting $\lambda_{\mathtt{sh}X}$ where λ_X is refined with sharing (we are vague about sharing on purpose) and evaluation in λ_X is simulated by some refinement $\to_{\mathtt{sh}X}$ of $\to_X$. The situation can then be refined as in the following diagram:

(1)

In the best cases [85, 13, 21] the simulations have bilinear overhead, that is, linear in the number of steps and in the size of the initial term. A term with sharing t represents the ordinary term $t{\downarrow}$ obtained by unfolding the sharing in t – the key point is that t can be exponentially smaller than $t{\downarrow}$. Evaluation in $\lambda_{\mathtt{sh}X}$ produces a shared normal form $\mathtt{nf}_{\mathtt{sh}X}(t)$ that is a compact representation of the ordinary result, that is, such that $\mathtt{nf}_{\mathtt{sh}X}(t){\downarrow} = \mathtt{nf}_X(t)$.

Let us stress that one needs sharing to obtain the simulations but then the strategy proved to be reasonable is the one without sharing (that is, $\to_X$) – here sharing is the key tool for the proof, but the cost model is not taken on the calculus with sharing.

Sharing and Reasonable Strategies. The kind of sharing at work in diagram (1), and therefore the definitions of $\lambda_{\mathtt{sh}X}$ and $\to_{\mathtt{sh}X}$, depends very much on the strategy $\to_X$. Let us fix some terminology.

The *weak* λ-calculus is the sub-calculus in which evaluation does not enter into abstractions, and, with the additional hypothesis that terms are *closed*, it models functional programming languages. The strong λ-calculus is the case where evaluation enters into function bodies, and its main domain of application are proof assistants.

The first result for weak strategies is due to Blelloch and Greiner in 1995 [32] and concerns weak CbV evaluation. Similar results were then proved again, independently, by Sands, Gustavsson, and Moran in 2002 [85] who also addressed CbN and CbNeed, and by combining the results by Dal Lago and Martini in 2009 in [41] and [40], who also addressed CbN in [61]. Actually already in 2006, Dal Lago and Martini proposed a reasonable cost model for the CbV case [39], but that cost model does not count 1 for each β-step.

Note, *en passant*, that *reasonable* does not mean *efficient*: CbN and CbV are incomparable for efficiency, and CbNeed is more efficient than CbN, and yet they are all reasonable. *reasonable* and *efficient* are indeed unrelated properties of strategies. Roughly, efficiency is a *comparative* property, it makes sense only if there are many strategies and one aims at comparing them. Being reasonable instead is a property of the strategy itself, independently of any other strategy, and it boils down to the fact that the strategy can be implemented with a negligible overhead.

In the strong case, at present, only one reasonable strategy is known, the leftmost-outermost (LO) strategy, that is the extension of the CbN strategy to the strong case. The result is due to Accattoli and Dal Lago [17] (2014), and it is inherently harder than the weak case, as a more sophisticated notion of sharing is required.

There also is an example of an unreasonable strategy. Asperti and Mairson in [27] (1998) proved that Lévy's parallel optimal evaluation [67] is unreasonable. Careful again: unreasonable does not mean inefficient (but it does not mean efficient either).

An Anecdote. To give an idea of the subtlety but also of how much these questions are neglected by the community, let us report an anecdote. In 2011, we attended a talk where the speaker started by motivating the study of strong evaluation as follows. There exists a family $\{s_n\}_{n\in\mathbb{N}}$ of terms such that s_n evaluates in $\Omega(2^n)$ steps with any weak strategy while it takes $\mathcal{O}(n)$ steps with rightmost-innermost strong evaluation. Thus, the speaker concluded, strong evaluation can be faster than weak evaluation, and it is worth studying it.

Such a reasoning is wrong (but the conclusion is correct, it is worth studying strong evaluation!). It is based on the hidden assumption that it makes sense to count the number of β-steps and compare strategies accordingly. Such an assumption, however, is valid only if the compared strategies are reasonable, that is, if it is proved that their number of steps is a reasonable time measure. In the talk, the speaker was comparing weak strategies, of which we have various reasonable examples, together with a strong strategy that is not known to be reasonable. In particular, rightmost-innermost evaluation is probably unreasonable, given that even when decomposed with sharing it lacks one of the key properties (the sub-term property) used in all proofs of reasonability in the literature.

That talk was given in front of an impressive audience, including many big names and historical figures of the λ-calculus. And yet no one noticed that identifying the number of β steps with the actual cost was naive and improper.

Apart from avoiding traps as the one of the anecdote, a proper approach to the cost of computation sheds a new light on questions of various nature. The next two subsections discuss the practical case of abstract machines and the theoretical case of denotational semantics.

2.1 Abstract Machines

The first natural research direction is the re-understanding of implementation techniques from a quantitative point of view.

Environment-Based Abstract Machines. The theory of implementations of λ-calculi is mainly based on environment-based abstract machines, that use *environments* to implement sharing and avoid size explosion. Having a reasonable cost model of reference, namely the number of β-steps of the strategy implemented by the machine, it is possible to bound the complexity of the machine overhead as a function of the size of the initial term and the number of steps taken by the strategy in the calculus.

A complexity-based approach to abstract machines is not a pedantic formality: Cregut's machine [37], that was the only known machine for strong evaluation for 25 years, has an exponential overhead with respect to the number of β-steps. Essentially, Cregut machine is as bad as implementing β-reduction *literally* as it is defined, without any form of sharing.

Similarly, the abstract machine for open terms described by Grégoire and Leroy in [50] suffers of exponential overhead, even if the authors in practice implement a slightly different machine with polynomial overhead.

In collaboration with Barenbaum, Mazza, Sacerdoti Coen, Guerrieri, Barras, and Condoluci, we developed a new theory of abstract machines [9, 10, 13, 7, 14, 11, 21, 15], where different term representations, data structures, and evaluation techniques are studied from a complexity point of view, compared, and sometimes improved to match a certain complexity. Some of the outcomes have been:

- A reasonable variant of Cregut's machine [7].
- A detailed study of how to improve Grégoire and Leroy's machine [13, 21].
- The first results showing that de Bruijn indices bring no asymptotic speed-up with respect to using names and perform α-renaming [11].
- The proof that administrative normal forms bring no asymptotic slowdown [15] (in contrast to what claimed by Kennedy in [52]).
- and, as a side contribution, the simplest presentations of CbNeed [9].

Apart from two exceptions actually focusing on other topics (the already cited studies by Blelloch and Greiner [32] and by Sands, Gustavsson, and Moran [85]), the literature before this new wave never studied the complexity of abstract machines.

Token-Based Abstract Machines. There is another class of abstract machines that is much less famous than those based on environments. These are so-called *token-based* abstract machines, introduced by Danos and Regnier [43], then studied by Mackie [69], Schöpp [86] and more recently by Mazza [72], Muroya and Ghica [78], and Dal Lago and coauthors [56, 57, 62, 58]. Their theoretical background is Girard's geometry of interaction. The basic idea is that, instead of storing all previously encountered β-redexes in environments, these machines keep a minimalistic amount of information inside a data structure called *token*, that is used to navigate the program without ever modifying it. The aim is to have an execution model that sacrifices time, by possibly repeating some work already done, in order to be efficient with respect to space. Various researchers conjecture the size of the token to be a reasonable cost model for space, but there are no results in the literature.

2.2 Denotational Semantics

Another research direction is the connection between the cost of computations and denotational semantics. At first sight, it looks like a non-topic, because denotational semantics are invariant under evaluation, and so it seems that they cannot be sensitive to intensional properties such as evaluation lengths. There are however hints that the question is subtler than it seems at first sight.

There are works in the literature trying to address somehow the question, by either building model of logics / λ-calculi with bounded complexity [77, 28, 59, 66], or by designing resource-sensitive models [48, 60, 63], or by extracting evaluation bounds from some semantics, typically game semantics [34, 35, 26]. The questions we propose here are related, but – at present – very open and somewhat vague. They stem from the work of Daniel de Carvalho on non-idempotent intersection types [44] (finally published in 2018 but first appeared in 2007), or, as we prefer to call them, *multi types* (because non-idempotent intersections can be seen as multi-sets).

One of de Carvalho's ideas is that even if the interpretation $[\![t]\!]$ of a λ-term t cannot tell us anything about the evaluation of t itself, there is still hope that (when t and s are normal) $[\![t]\!]$ and $[\![s]\!]$ provide information about the evaluation of ts, typically about the number of steps to normal form and the size of the normal form. De Carvalho studies the CbN relational model (induced by the relational model of linear logic via the call-by-name translation of λ-calculus into linear logic), that can be syntactically presented via multi types. The key points of his study are:

1. the size of a type derivation π of $\Gamma \vdash t : A$ provides bounds to both the number of CbN steps to evaluate t to its normal form $\mathtt{nf}(t)$ and the size $|\mathtt{nf}(t)|$ of $\mathtt{nf}(t)$.
2. the size of the types themselves – more precisely A plus the types in the typing context Γ – bounds $|\mathtt{nf}(t)|$.
3. the interpretation of a term in the model is the set of type judgements – again, A plus the types in the typing context Γ – that can be derived for it in the typing system, that is,

   $$[\![t]\!] := \{((M_1, \ldots, M_n), A) \mid x_1 : M_1, \ldots, x_n : M_n \vdash t : A\}.$$

   where the M_i are multi-sets of types.
4. Minimal derivations provide the exact measure of the number of steps *plus* the size of the normal form. Similarly the minimal derivable types provide the exact measure of the normal form.
5. From $[\![t]\!]$ and $[\![s]\!]$ one can bound the number of steps *plus* the size of the normal form of ts.

Such a strong correspondence does not happen by chance. Further work [45, 19] has shown that multi types and the relational model compute according to natural strategies in linear logic proof nets, including in particular CbN evaluation. The link is natural: multi types are intersection types without idempotency, that is, without the principle $A \cap A = A$, or, said differently, the number of times that A is appear does matter... exactly as in linear logic. Similar results connect strategies in linear logic proof nets and game semantics [42, 34, 35].

The Extraction of Computational Mechanisms from Models. De Carvalho's work suggests that denotational models hide a computational machinery behind their compositional principles. The one between relational and game semantics and evaluation strategies may be only the easiest one to observe. A natural question is: what about (idempotent) intersection types? They are syntactic presentations of domain-based semantics *à la Scott*. Despite being the first discovered model of the λ-calculus, nothing is known about their hidden evaluation

scheme, apart from the fact that it is not the one behind the relational model, for which non-idempotency is required. Intuition says that idempotency may model some form of sharing. The question is however open.

Models Internalising Sharing. The key points about size explosion and reasonable cost models are:

1. In an evaluation to normal form $t \to_\beta^n \mathtt{nf}(t)$ the size $|\mathtt{nf}(t)|$ of the normal form may be exponential in n;
2. n is nonetheless a reasonable measure of complexity (if evaluation is done according to a reasonable strategy);
3. this is possible because sharing allows to compute a compact representation $\mathtt{nf}_{\mathtt{sh}X}(t)$ of the normal form that is polynomial in n.

Now, the interpretation of a term t in de Carvalho's CbN relational model is a set whose smallest element is as large as the normal form $\mathtt{nf}(t)$. It seems then difficult that such a model may give accurate information about the time cost of λ-terms, as in general from $[\![t]\!]$ and $[\![s]\!]$ one can only obtain information about the evaluation of ts *together* with the size of $\mathtt{nf}(ts)$, which may however be much larger than the length of evaluation to normal form.

For such a reason, Accattoli, Graham-Lengrand, and Kesner in [19] refined de Carvalho's type system as to have type derivations that provide separate bounds with respect to evaluation lengths and the size of normal forms. The idea is that judgements now have the form:

$$x_1 : M_1, \ldots, x_n : M_n \vdash^{(b,r)} t : A$$

where b provides a bound to number of β-steps to evaluate t to its normal form $\mathtt{nf}(t)$ (that is, the evaluation length) and r is a bound to the size of $\mathtt{nf}(t)$. This is a slight improvement, but still not enough, as such a refined information is on type derivations but not on the types themselves, that are what defines the semantical interpretation.

An important open question is then whether there are models where the (smallest point in the) interpretation of t is of the order of the size of the compact representation $\mathtt{nf}_{\mathtt{sh}X}(t)$ of the normal form, and not of the order of $|\mathtt{nf}(t)|$. Roughly, it would be a model whose hidden computational mechanism uses sharing as it is needed to obtain reasonable implementations.

We believe that this is an important question to answer in order to establish a semantical understanding of sharing and close the gap between semantical and syntactical studies.

We conjecture that the CbV relational model may have this property, as – despite being built from non-idempotent intersection types – it allows a special use of the empty intersection, which is the only idempotent type of the model (as the intersection of two empty intersections is still empty) and that may be the key tool to internalise sharing.

Lax and Tight Models with Respect to Strategies. A related question is how to refine the notion of model as to be relative to an evaluation strategy. The need for a refined notion of model arises naturally when studying CbNeed evaluation. CbNeed is sometimes considered simply as an optimisation of CbN. It is however better understood as an evaluation scheme on its own, obtained by mixing the good aspects of both CbN and CbV, and observationally equivalent to CbN. Because of such an equivalence, every model of CbN provides a model of CbNeed. In particular, the relational model built on multi types discussed above is a model of CbNeed – Kesner used it to provide a simple proof of the equivalence of CbN and CbNeed [53].

The bounds provided by that relational model, however, are not exact for CbNeed, since they are exact for CbN, and thus cannot capture its faster alternative. Recently, Accattoli, Guerrieri, and Leberle have obtained a multi type system providing exact bounds for CbNeed [23]. The type system induces a model, which is "better" than de Carvalho's one, as it more precisely captures CbNeed evaluation lengths. There is however no abstract, categorical way – at present – to separate the two models, as there are no abstract notions of lax or tight model with respect to an evaluation strategy. To be fair, there is not even a notion of categorical model of CbNeed, that should certainly be developed.

3 From the λ-calculus to λ-calculi

There are at least two theories of the λ-calculus, the strong and the weak. Historically, the theory of λ-calculus rather dealt with *strong* evaluation, and it is only since the seminal work of Abramsky and Ong [2] that the theory took weak evaluation seriously. Dually, the practice of functional languages mostly ignored *strong* evaluation, with the notable exception of Crégut [36, 37] (1990) and, more recently, the semi-strong approach of Grégoire and Leroy [50] (2002), following the idea that a function is an algorithm to apply to some data rather than data by itself. Strong evaluation is nonetheless essential in the implementation of proof assistants or higher-order logic programming, typically for type-checking in frameworks with dependent types as the Edinburgh Logical Framework or the Calculus of Constructions, as well as for unification modulo $\beta\eta$ in simply typed frameworks like λ-prolog.

There is also another axis of duplication of work. Historically, the theory is mostly studied with respect to CbN evaluation, while functional programming languages tend to employ CbV (of which there are no traces in Barendregt's and Krivine's books) or CbNeed. The differences between these settings is striking. What is considered *the* λ-calculus is the strong CbN λ-calculus, and it has been studied in depth, despite the fact that no programming language or proof assistant implements it. Given the practical relevance of weak CbV with closed terms, that is the backbone of the languages of the ML family, such a setting is also well known, even if its theory is anyway less developed than the one for strong CbN. Weak CbN is also reasonably well studied. But as soon as one steps out of these settings the situation changes drastically. The simple extension of weak CbV to open terms is already a delicate subject, not to speak of strong CbV. About CbNeed– that is the strategy implemented by Haskell – the situation is even worse, as its logical (Curry-Howard) understanding is less satisfactory than for CbN and CbV, its semantical understanding essentially inexistent, and there is only one paper about Strong CbNeed, by Balabonski et al. [29].

We believe that there is a strong need of reconciling theory and practice. A key step has to be a change of perspective. The λ-calculus comes in different flavors, and in my experience a comparative study is very fruitful, as it identifies commonalities, cleaning up concepts, and also stresses differences and peculiar traits of each setting.

The Open Setting. There actually is an intermediate setting between the weak and the strong λ-calculi. First of all, it is important to stress that weak evaluation is usually paired with the hypothesis that terms are *closed*, which is essential in order to obtain the key property that weak normal forms are all and only abstractions – this is why we rather prefer to call it the *closed λ-calculus*. On the other hand, in the strong case it does not make sense to restrict to closed terms, because evaluation enters into function bodies, that cannot be assumed closed. Such a difference forbids to see the strong case as the iteration of the closed one under abstraction, as the closed hypothesis is not stable under iterations.

It is then natural to consider the case of weak evaluation with open terms, what we like to refer to as the *open λ-calculus*. The open λ-calculus can be iterated under abstraction providing a procedure for strong evaluation – this is for instance done by Grégoire and Leroy in [50]. Historically, the open case was neglected. The reason is that the differences between the closed and open case are striking in CbV but negligible in CbN. Since the CbV literature focused on the closed setting, the open case sat in a blind spot.

The study of reasonable cost models made evident that different, increasingly sophisticated techniques are needed in the three settings – closed, open, and strong – in order to obtain reasonable abstract machines. Moreover, such a classification is stable by moving on the other axis, that is, closed CbN, closed CbV, and closed CbNeed share the same issues, and similarly for the three open cases and the three strong cases.

3.1 Open Call-by-Value

The ordinary approach to CbV, due to Plotkin, has a famous property that we like to call *harmony*: a closed term either is a value or it CbV reduces.

The Issue with Open Terms. Plotkin's operational semantics for CbV does have some good properties on open terms. For instance, it is confluent and it admits a standardisation theorem, as Plotkin himself proved [81]. It comes however also with deep problems. First, open terms bring *stuck β-redexes* such as

$$(\lambda x.t)(yz)$$

the argument is not a value and will never become one, thus the term is CbV normal (there is a β-redex but it is not a β_v-redex) – that is, harmony is lost.

Unfortunately, stuck redexes induce a further problem: they block creations. Consider the open term (where δ is the usual duplicator)

$$((\lambda x.\delta)(yz))\delta$$

As before, it is normal in Plotkin's traditional λ-calculus for CbV. Now, however, there are semantic reasons to consider it as a divergent term. Roughly, there are denotational semantics that are *adequate* on closed terms (adequacy means that the semantical interpretation of a term t is non-empty if and only if t evaluates to a value; by harmony, it is equivalent to say that t is *not* divergent) and with respect to which $(\lambda x.\delta)(yz)\delta$ has empty interpretation (that is, it is considered as being a divergent term, while it is normal). This semantical mismatch has first been pointed out by Paolini and Ronchi Della Rocca [80, 79, 82]. A similar phenomenon happens if one looks at the interpretation of the term according to the CbV translation into linear logic, as pointed out by the author in [5]. Essentially, that term is expected to reduce as follows

$$((\lambda x.\delta)(yz))\delta \to \delta\delta$$

and *create* the redex $\delta\delta$. Evaluation then would go on forever. If one sticks to Plotkin's rewriting rule, however, the creation is blocked by the stuck redex $(\lambda x.\delta)(yz)$ and the term is normal – quite a mismatch with what is expected semantically and logically. A similar problem affects the term $\delta((\lambda x.\delta)(yz))$ that is also normal while it should be divergent.

The subtlety of the problem is that one would like to have a notion of CbV evaluation on open terms making terms such as $((\lambda x.\delta)(yz))\delta$ and $\delta((\lambda x.\delta)(yz))$ divergent, thus extending Plotkin's evaluation, but at the same time preserving CbV divergence without collapsing on CbN, that is, such that $(\lambda x.y)(\delta\delta)$ has no normal form.

Open CbV. In his seminal work, Plotkin already pointed out an asymmetry between CbN and CbV: his continuation-passing style (CPS) translation is sound and complete for CbN, but only sound for CbV. This fact led to a number of studies about monad, CPS, and logical translations [76, 83, 84, 70, 46, 51] that introduced many proposals of improved calculi for CbV. The dissonance between open terms and CbV has been repeatedly pointed out and studied *per se* via various calculi related to linear logic [24, 5, 33, 13]. To improve the implementation of the Coq proof assistant, Grégoire and Leroy introduced another extension of CbV to open terms [50]. A further point of view on CbV comes from the computational interpretation of sequent calculus due to Curien and Herbelin [38]. An important point is that most of these works focus on strong CbV.

Inspired by the robustness of complexity classes via reasonable cost models, in a series of works Accattoli, Guerrieri and Sacerdoti Coen define what they call *open CbV*, that is the isolation of the case of weak evaluation with open terms (rather than strong evaluation) that they show to be a simpler abstract framework with many different incarnations:

- *Operational semantics*: in [20], 4 representative calculi extending Plotkin's CbV are compared, and showed to be termination equivalent (the evaluation of a fixed term t terminates in one of the calculi if and only if terminates in the others). Moreover, evaluation lengths (in 3 of them) are linearly related;
- *Cost model*: In [13, 21], such a common evaluation length is proved to be a reasonable cost model, by providing abstract machines that are proved reasonable;
- *Denotational semantics*: In [22], Ehrhard's relational model of CbV [47] is shown to be an adequate denotational model of open CbV, providing exact bounds along the lines of de Carvalho's work.

Last, let us stress both the termination equivalence of the 4 presentations of open CbV and the relationship with the denotational semantics make crucial use of formalisms with sharing.

A key point is that each presentation of open CbV has its pros and cons, but none of them is perfect. This is in contrast to CbN, where there are no doubts about the canonicity of its presentation.

Strong CbV. The obvious next step is to lift the obtained relationships to strong evaluation. This is a technically demanding ongoing work.

3.2 Benchmark for λ-Calculi

The work on open CbV forced to ask ourselves what are the guiding principles that define *a good λ-calculus*. This is especially important if one takes seriously the idea that there is not just one λ-calculus but a rich set of λ-calculi, in order to fix standards and allow comparisons.

It is of course impossible to give an absolute answer, because different applications value different properties. It is nonetheless possible to collect requirements that seem desirable in order to have an abstract framework that is also useful in practice. We can isolate at least six principles to be satisfied by a good λ-calculus:

1. *Rewriting*: there should be a small-step operational semantics having nice rewriting properties. Typically, the calculus should be non-deterministic but confluent, and a deterministic evaluation strategy should emerge naturally from some good rewriting property (factorisation / standardisation theorem, or the diamond property). The *strategy emerging from the calculus* principle guarantees that the chosen evaluation is not ad-hoc.
2. *Logic*: typed versions of the calculus should be in Curry-Howard correspondences with some proof systems, providing logical intuitions and guiding principles for the features of the calculus and the study of its properties.

3. *Implementation*: there should be a good understanding of how to decompose evaluation in micro-steps, that is, at the level of abstract machines, in order to guide the design of languages or proof assistants based on the calculus.
4. *Cost model*: the number of steps of the deterministic evaluation strategy should be a reasonable time cost model, so that cost analyses of λ-terms are possible and independent of implementative choices.
5. *Denotations*: there should be denotational semantics that reflect some of its properties, typically an adequate semantics reflecting termination. Well-behaved denotations guarantee that the calculus is somewhat independent from its own syntax, which is a further guarantee that it is not ad-hoc.
6. *Equality*: contextual equivalence can be characterised by some form of bisimilarity, showing that there is a robust notion of program equivalence. Program equivalence is indeed essential for studying program transformations and optimisations at work in compilers.

Finally, there is a sort of meta-principle: the more principles are connected, the better. For instance, it is desirable that evaluation in the calculus correspond to cut-elimination in some logical interpretation of the calculus. Denotations are usually at least required to be *adequate* with respect to the rewriting: the denotation of a term is non-degenerate if and only if its evaluation terminates. Additionally, denotations are *fully abstract* if they reflect contextual equivalence. And implementations have to work within an overhead that respects the intended cost semantics. Ideally, all principles are satisfied and perfectly interconnected.

Of course, some specific cases may drop some requirements – for instance, a probabilistic λ-calculus would not be confluent – some properties may also be strengthened – for instance, equality may be characterised via a separation theorem akin to Böhm's – and other principles may be added – categorical semantics, graphical representations, etc. As concrete cases, the strong CbN λ-calculus satisfies all these principles, while at present open CbV satisfies the first 5, with an high degree of connection between them, strong CbNeed only 2 or 3 of them, and strong CbV none of them.

We are here exposing these principles hoping to receive feedback from the community, for instance, helping us identifying further essential principles, if any. We also believe that single researchers tend to specialise excessively in one of the principles, forgetting the global picture, which is instead where the meaning of the single studies stems, in our opinion. A new book or new introductory notes on the λ-calculus should be developed around the idea of connecting these principles, to form students in the field.

4 Sharing

The aim of this section is to convince the reader that sharing is an unavoidable ingredient of a modern understanding of λ-calculi. This is done by pointing out a number of reasons, including a historical perspective, and giving some examples coming from the work of the author. In particular, we would like to stress that, rather than a feature such as continuations or pattern matching that can be added on top of λ-calculi, sharing is the *éminence grise* of the higher-order world.

Sharing is a vague term that means different things in different contexts. For instance, sharing as in environment-based abstract machines or sharing as in Lamping's sharing graphs [64] implementing Lévy's parallel optimal strategy [67] have not much in common.

Here we refer to the simplest possible form, that is closer to sharing as it appears in abstract machines. While we do want to commit to a certain style, as it shall be evident below, we also want to stay vague about it, as such a style can be realized in various ways.

The simplest construct for sharing is a `let` $x = s$ `in` t expression, that is a syntactic annotation for t *where* x *will be substituted by* s. We also write it more concisely as $t[x{\leftarrow}s]$ (not to be confused with meta-level substitution, noted $t\{x{\leftarrow}s\}$) and call it ES (for *explicit sharing*, or *explicit substitution*). Thanks to ES, β-reduction can be decomposed into more atomic steps. The simplest decomposition splits β-reduction as follows:

$$(\lambda x.t)s \quad \rightarrow_{\mathsf{B}} \quad t[x{\leftarrow}s] \quad \rightarrow_{\mathsf{ES}} \quad t\{x{\leftarrow}s\}$$

It is well-known that ES are somewhat redundant, as they can always be removed, by simply coding them as β-redexes. They are however more than syntactic sugar, as they provide a simple and yet remarkably effective tool to understand, implement, and program with λ-calculi and functional programming languages:

- From a logical point of view ES are the proof terms corresponding to the extension of natural deduction with a cut rule, and the cut rule is *the* rule representing computation, according to Curry-Howard.
- From an operational semantics point of view, they allow elegant formulations of subtle strategies such as call-by-need evaluation – various presentations of call-by-need use ES [89, 65, 71, 25, 87, 55] and a particularly simple one is in [9].
- From a programming point of view, `let` expressions are part of the syntax of all functional programming languages we are aware of.
- From a rewriting point of view, they enable proof techniques that are not available within the λ-calculus, as we are going to explain below.
- Finally, sharing is used in all implementations of tools based on the λ-calculus to circumvent *size explosion.*

A Historical Perspective. Between the end of the eighties and the beginning of the nineties, three independent decompositions of the λ-calculus arose with different aims and techniques:

1. Girard's *linear logic* [49], where the λ-calculus is decomposed in two layers, *multiplicative* and *exponential*;
2. Abadi, Cardelli, Curien, and Lévy's *explicit substitutions* [1], that are refinements of the λ-calculus where meta-level substitution is delayed, by introducing explicit annotations, and then computed in a micro-step fashion.
3. Milner, Parrow, and Walker's π*-calculus* [75], where the λ-calculus can be represented, as shown by Milner [73], by decomposing evaluation into message passing and process replication.

All these settings introduce an explicit treatment of sharing – called *exponentials* in linear logic, or explicit substitutions, or *replication* in the π-calculus. At first sight these approaches look quite different. It took more than 20 years to obtain the *Linear Substitution Calculus* (LSC), a λ-calculus with sharing that captures the essence of all three approaches in a simple and manageable formalism.

The LSC [3, 12] is a refinement of the λ-calculus with sharing, introduced by Accattoli and Kesner as a minor variation over a calculus by Milner [74]. The rest of the section is a gentle introduction to some of its unique features. We shall also discuss an even simpler setting, the *substitution calculus*, that is probably the most basic setting for sharing.

4.1 Rewriting at a Distance, or Disentangling Search and Substitution

Usually, λ-calculi with ES have rules accounting for two tasks, one is decomposing (or just executing) the substitution process and one is commuting ES with other constructs. For the time being, let us simplify the first task, and assume that ES are executed in just one step, as follows

$$t[x{\leftarrow}s] \quad \rightarrow_{\mathsf{ES}} \quad t\{x{\leftarrow}s\}$$

The second task is required for instance in situations such as

$$(\lambda x.t)[y{\leftarrow}s]u$$

where the ES $[y{\leftarrow}s]$ is blocking the possibility of reducing the β redex $(\lambda x.t)u$. Usually the calculus is endowed with one of the two following rules

$$(\lambda x.t)[y{\leftarrow}s] \rightarrow_{\lambda} \lambda x.t[y{\leftarrow}s] \qquad\qquad t[y{\leftarrow}s]u \rightarrow_{@} (tu)[y{\leftarrow}s]$$

that incarnate opposite approaches to expose the β-redex and continue the computation.

There is however a simpler alternative. Instead of adding a rule to commute ES, one can generalise the notion of β-redex, by allowing the abstraction and the argument to interact *at a distance*, that is, even if there are ES in between:

$$(\lambda x.t)[y_1{\leftarrow}s_1]\ldots[y_k{\leftarrow}s_k]u \quad \rightarrow_{\mathsf{dB}} \quad t[x{\leftarrow}u][y_1{\leftarrow}s_1]\ldots[y_k{\leftarrow}s_k]$$

The name of the rule stands for *distant* $\mathtt{B}$, where $\mathtt{B}$ is the variant of β that creates an ES.

The rule can be made compact by employing *contexts*. Define *substitution contexts* as follows

$$L ::= \langle\cdot\rangle \mid L[x{\leftarrow}t]$$

Then the previous rule can be rewritten as:

$$L\langle\lambda x.t\rangle u \rightarrow_{\mathsf{dB}} L\langle t[x{\leftarrow}u]\rangle$$

Define the *substitution calculus* as the language of the λ-calculus with ES plus rules $\rightarrow_{\mathsf{dB}}$ and $\rightarrow_{\mathsf{ES}}$. Such a simple formalism already provides relevant insights.

First of all, it has a strong connection with the linear logic proof nets representation of the λ-calculus [8]. Rules $\rightarrow_{\mathsf{dB}}$ and $\rightarrow_{\mathsf{ES}}$ indeed correspond *exactly* to multiplicative and exponential cut-elimination in such a representation (if one assumes a *one shot* cut-elimination rule for the exponentials), where *exactly* means that there is a bijection of redexes providing a *strong* bisimulation between terms and proof nets (of that fragment). This happens because the *trick* of rewriting at a distance captures exactly the graphical dynamics of proof nets. Phrased differently, commuting rules such as $\rightarrow_{\lambda}$ or $\rightarrow_{@}$ have no analogous on proof nets.

4.2 A Rewriting Pearl: Confluence from Local Diagrams

The substitution calculus already allows to show that sharing enables simple and elegant proof techniques that are impossible in λ-calculi without sharing.

The best example concerns confluence. In rewriting, termination often allows to lift local properties to the global level. The typical example is Newman lemma, that in presence of strong normalisation lifts local confluence to confluence. The λ-calculus is locally confluent but not strongly normalising (not even weakly!), so Newman lemma cannot be applied. It is then necessary, for instance, to introduce parallel steps and adopt the Tait and Martin Löf proof technique.

In the substitution calculus, instead, we can prove confluence from local confluence. Newman lemma does not apply directly, but an elegant alternative reasoning is possible.

The key observation is that having decomposed β in two rules $\to_{\mathsf{dB}}$ and $\to_{\mathsf{ES}}$, one still has that the whole calculus may not terminate, but a new *local* termination principle is available: $\to_{\mathsf{dB}}$ and $\to_{\mathsf{ES}}$ are strongly normalising when considered separately. Local termination can be seen as the internalisation of a classic result, the *finite developments theorem*, stating that in the λ-calculus every evaluation that does not reduce redexes created by the sequence itself terminates. The proof of confluence of the substitution calculus goes as follows:

- Rules $\to_{\mathsf{dB}}$ and $\to_{\mathsf{ES}}$ are both locally confluent, so that by Newman lemma they are (separately) confluent.
- To obtain confluence of the full calculus we need another classic rewriting lemma by Hindley and Rosen: the union of confluent and commuting relations is confluent.
- We then need to prove commutation of $\to_{\mathsf{dB}}$ and $\to_{\mathsf{ES}}$, that is, if $s \;{}_{\mathsf{ES}}^{*}\!\!\leftarrow t \to_{\mathsf{dB}}^{*} u$ then there exists r such that $s \to_{\mathsf{dB}}^{*} r \;{}_{\mathsf{ES}}^{*}\!\!\leftarrow u$.
- Again, commutation is a global property that can be obtained via a local one. In this case there are no lemmas analogous to Newman (commutation is more general than confluence), but the fact that $\to_{\mathsf{dB}}$ cannot duplicate $\to_{\mathsf{ES}}$ implies that their local commutation diagram trivially lifts to global commutation.
- Then $\to_{\mathsf{dB}} \cup \to_{\mathsf{ES}}$ is confluent.

This proof scheme was used for the first time by Accattoli and Paolini in [24]. In [3], Accattoli uses the local termination principle to prove the head factorisation theorem for the LSC in a way that is impossible in the λ-calculus.

The moral is that introducing sharing enables rewriting proof techniques that are impossible without it.

4.3 Linear Substitutions, Weak Evaluation, and Garbage Collection

Let's now decompose the substitution rule and define the LSC. Most of the literature of ES decomposes the substitution process using rules that are entangled with commuting rules such as $\to_{\lambda}$ and $\to_{@}$ described above. The special ingredient of the LSC is that substitution is decomposed but also disentangled from the commuting process. To properly define the rules we need to introduce a general notion of context:

$$\text{ES CONTEXTS} \qquad C, D \;::=\; \langle\cdot\rangle \mid \lambda x.C \mid Ct \mid tC \mid C[x{\leftarrow}t] \mid t[x{\leftarrow}C]$$

Now, there are only two rules for evaluating explicit substitutions at a distance (plus dB, to create them):

$$\begin{array}{lrcll} \text{LINEAR SUBSTITUTION} & C\langle\!\langle x\rangle\!\rangle[x{\leftarrow}s] & \to_{\mathtt{ls}} & C\langle\!\langle t\rangle\!\rangle[x{\leftarrow}s] & \\ \text{GARBAGE COLLECTION} & t[x{\leftarrow}s] & \to_{\mathtt{gc}} & t & \text{if } x \notin \mathtt{fv}(t) \end{array}$$

The idea is that given $t[x{\leftarrow}s]$ either x has no occurrences in t, and so the garbage collection rule $\to_{\mathtt{gc}}$ applies, or x does occur in t, which can then be written as $C\langle x\rangle$ for some context C (not capturing x), potentially in more than one way. The linear substitution rule then allows to replace the occurrence of x isolated by C without moving the ES $[x{\leftarrow}s]$.

The LSC is given by rules $\to_{\mathtt{dB}}$, $\to_{\mathtt{ls}}$, and $\to_{\mathtt{gc}}$. More precisely, the definition of these rules includes a further contextual closure (as it is standard also in the λ-calculus), that is, one defines $C\langle t\rangle \to_{\mathtt{dB}} C\langle s\rangle$ if $t \to_{\mathtt{dB}} s$, and similarly for the other rules.

Confluence for the LSC can be proved following exactly the same schema used for the substitution calculus.

Confluence of Weak Evaluation. One of the nice features of the LSC is that its definition is parametric in the grammar of contexts. For instance, we can define the Weak LSC by simply restricting general contexts C (used both to define rule $\rightarrow_{\mathtt{ls}}$ at top level and to give the contextual closure of the three rules) to weak contexts W that do not enter into abstractions:

$$\text{WEAK ES CONTEXTS} \qquad W, W' \;::=\; \langle\cdot\rangle \mid Wt \mid tW \mid W[x{\leftarrow}t] \mid t[x{\leftarrow}W]$$

The Weak LSC is confluent, and the proof goes always along the same lines.

Note that this is in striking contrast to what happens in the λ-calculus. Defining the weak λ-calculus by simply restricting the contextual closure to weak contexts produces a non-confluent calculus. For instance, the following diagram cannot be closed:

$$(\lambda x.\lambda y.yx)I \;{}_{\beta}{\leftarrow}\; (\lambda x.\lambda y.yx)(II) \rightarrow_{\beta} \lambda y.(y(II))$$

because II in the right reduct occurs under abstraction and it is then frozen. There are solutions to this issue, see Lévy and Maranget's [68], but they are all ad-hoc. In the LSC, instead, everything is as natural as it can possibly be.

Garbage Collection. Another natural property that holds in the LSC but not in the λ-calculus is the postponement of garbage collection. Rule $\rightarrow_{\mathtt{gc}}$ can indeed be postponed, that is, one has that if $t \rightarrow^*_{LSC} s$ then $t \rightarrow^*_{\mathtt{dB},\mathtt{ls}}\rightarrow^*_{\mathtt{gc}} s$. Since $\rightarrow_{\mathtt{gc}}$ is also strongly normalising, it is then *safe* to remove it and only consider the two other rules, $\rightarrow_{\mathtt{dB}}$ and $\rightarrow_{\mathtt{ls}}$.

The postponement of garbage collection models the fact that in programming languages the garbage collector acts asynchronously with respect to the execution flow.

In the λ-calculus, *erasing steps* are the analogous of garbage collection. A step $(\lambda x.t)s \rightarrow_{\beta} t\{x{\leftarrow}s\}$ is erasing if – like for garbage collection – the abstracted variable x does not occur in t, so that $t\{x{\leftarrow}s\} = t$, and s is simply erased.

The key point is that erasing steps cannot be postponed in the λ-calculus. Consider indeed the following sequence:

$$(\lambda x.\lambda y.y)ts \rightarrow_{\beta} (\lambda y.y)s \rightarrow_{\beta} s$$

The first step is erasing but it *cannot* be postponed after the second one, because the second step is *created* by the first one.

A Bird's Eye View. The list of unique elegant properties of the LSC is long. For instance, it has deep and simple connections to linear logic proof nets [8], the π-calculus [4], abstract machines and CbNeed [9], reasonable cost models [16, 18], open CbV [20], linear head reduction [3], multi types [19], it admits a residual system and a rich theory of standardisation [12], and even Lévy optimality [30]. Essentially, it is the canonical decomposition of the λ-calculus, and, in many respects, it is more expressive and flexible than the λ-calculus – it is a sort of λ-calculus 2.0.

Should then the LSC replace the λ-calculus? No. The point is not which system is the best. The point is acknowledging that the field is rich, and that even the simplest higher-order framework declines itself in a multitude of ways (weak/open/strong, cbn/cbv/cbneed, no sharing/one shot sharing/linear sharing), each one with its features and being a piece of a great puzzle.

References

1. Martín Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy. Explicit Substitutions. *J. Funct. Program.*, 1(4):375–416, 1991. doi:10.1017/S0956796800000186.
2. Samson Abramsky and C.-H. Luke Ong. Full Abstraction in the Lazy Lambda Calculus. *Inf. Comput.*, 105(2):159–267, 1993.
3. Beniamino Accattoli. An Abstract Factorization Theorem for Explicit Substitutions. In *RTA*, pages 6–21, 2012. doi:10.4230/LIPIcs.RTA.2012.6.
4. Beniamino Accattoli. Evaluating functions as processes. In *TERMGRAPH*, pages 41–55, 2013. doi:10.4204/EPTCS.110.6.
5. Beniamino Accattoli. Proof nets and the call-by-value λ-calculus. *Theor. Comput. Sci.*, 606:2–24, 2015. doi:10.1016/j.tcs.2015.08.006.
6. Beniamino Accattoli. The Complexity of Abstract Machines. In *WPTE@FSCD 2016*, pages 1–15, 2016. doi:10.4204/EPTCS.235.1.
7. Beniamino Accattoli. The Useful MAM, a Reasonable Implementation of the Strong λ-Calculus. In *WoLLIC 2016*, pages 1–21, 2016. doi:10.1007/978-3-662-52921-8_1.
8. Beniamino Accattoli. Proof Nets and the Linear Substitution Calculus. In *ICTAC 2018*, pages 37–61, 2018. doi:10.1007/978-3-030-02508-3_3.
9. Beniamino Accattoli, Pablo Barenbaum, and Damiano Mazza. Distilling abstract machines. In *ICFP 2014*, pages 363–376, 2014. doi:10.1145/2628136.2628154.
10. Beniamino Accattoli, Pablo Barenbaum, and Damiano Mazza. A Strong Distillery. In *APLAS 2015*, pages 231–250, 2015. doi:10.1007/978-3-319-26529-2_13.
11. Beniamino Accattoli and Bruno Barras. Environments and the complexity of abstract machines. In *PPDP 2017*, pages 4–16, 2017. doi:10.1145/3131851.3131855.
12. Beniamino Accattoli, Eduardo Bonelli, Delia Kesner, and Carlos Lombardi. A nonstandard standardization theorem. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 659–670, 2014. doi:10.1145/2535838.2535886.
13. Beniamino Accattoli and Claudio Sacerdoti Coen. On the Relative Usefulness of Fireballs. In *LICS 2015*, pages 141–155, 2015. doi:10.1109/LICS.2015.23.
14. Beniamino Accattoli and Claudio Sacerdoti Coen. On the value of variables. *Inf. Comput.*, 255:224–242, 2017. doi:10.1016/j.ic.2017.01.003.
15. Beniamino Accattoli, Andrea Condoluci, Giulio Guerrieri, and Claudio Sacerdoti Coen. Crumbling Abstract Machines. Submitted, 2019.
16. Beniamino Accattoli and Ugo Dal Lago. On the Invariance of the Unitary Cost Model for Head Reduction. In *RTA*, pages 22–37, 2012. doi:10.4230/LIPIcs.RTA.2012.22.
17. Beniamino Accattoli and Ugo Dal Lago. Beta reduction is invariant, indeed. In *CSL-LICS '14*, pages 8:1–8:10, 2014. doi:10.1145/2603088.2603105.
18. Beniamino Accattoli and Ugo Dal Lago. (Leftmost-Outermost) Beta-Reduction is Invariant, Indeed. *Logical Methods in Computer Science*, 12(1), 2016. doi:10.2168/LMCS-12(1:4)2016.
19. Beniamino Accattoli, Stéphane Graham-Lengrand, and Delia Kesner. Tight typings and split bounds. *PACMPL*, 2(ICFP):94:1–94:30, 2018. doi:10.1145/3236789.
20. Beniamino Accattoli and Giulio Guerrieri. Open Call-by-Value. In *APLAS 2016*, pages 206–226, 2016. doi:10.1007/978-3-319-47958-3_12.
21. Beniamino Accattoli and Giulio Guerrieri. Implementing Open Call-by-Value. In *FSEN 2017, Tehran, Iran, April 26-28, 2017, Revised Selected Papers*, pages 1–19, 2017. doi:10.1007/978-3-319-68972-2_1.
22. Beniamino Accattoli and Giulio Guerrieri. Types of Fireballs. In *Programming Languages and Systems - 16th Asian Symposium, APLAS 2018, Wellington, New Zealand, December 2-6, 2018, Proceedings*, pages 45–66, 2018. doi:10.1007/978-3-030-02768-1_3.
23. Beniamino Accattoli, Giulio Guerrieri, and Maico Leberle. Types by need. In *Programming Languages and Systems - 28th European Symposium on Programming, ESOP 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS*

2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, pages 410–439, 2019. doi:10.1007/978-3-030-17184-1_15.

24 Beniamino Accattoli and Luca Paolini. Call-by-Value Solvability, revisited. In *FLOPS*, pages 4–16, 2012. doi:10.1007/978-3-642-29822-6_4.

25 Zena M. Ariola, Matthias Felleisen, John Maraist, Martin Odersky, and Philip Wadler. The Call-by-Need Lambda Calculus. In *POPL'95*, pages 233–246, 1995. doi:10.1145/199448.199507.

26 Federico Aschieri. Game Semantics and the Geometry of Backtracking: a New Complexity Analysis of Interaction. *J. Symb. Log.*, 82(2):672–708, 2017. doi:10.1017/jsl.2016.48.

27 Andrea Asperti and Harry G. Mairson. Parallel Beta Reduction is not Elementary Recursive. In *POPL*, pages 303–315, 1998. doi:10.1145/268946.268971.

28 Patrick Baillot. Stratified coherence spaces: a denotational semantics for light linear logic. *Theor. Comput. Sci.*, 318(1-2):29–55, 2004. doi:10.1016/j.tcs.2003.10.015.

29 Thibaut Balabonski, Pablo Barenbaum, Eduardo Bonelli, and Delia Kesner. Foundations of strong call by need. *PACMPL*, 1(ICFP):20:1–20:29, 2017. doi:10.1145/3110264.

30 Pablo Barenbaum and Eduardo Bonelli. Optimality and the Linear Substitution Calculus. In *FSCD 2017*, pages 9:1–9:16, 2017. doi:10.4230/LIPIcs.FSCD.2017.9.

31 Hendrik Pieter Barendregt. *The Lambda Calculus – Its Syntax and Semantics*, volume 103 of *Studies in logic and the foundations of mathematics*. North-Holland, 1984.

32 Guy E. Blelloch and John Greiner. Parallelism in Sequential Functional Languages. In *FPCA*, pages 226–237, 1995. doi:10.1145/224164.224210.

33 Alberto Carraro and Giulio Guerrieri. A Semantical and Operational Account of Call-by-Value Solvability. In *Foundations of Software Science and Computation Structures - 17th International Conference, FOSSACS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, pages 103–118, 2014. doi:10.1007/978-3-642-54830-7_7.

34 Pierre Clairambault. Estimation of the Length of Interactions in Arena Game Semantics. In *FOSSACS*, pages 335–349, 2011. doi:10.1007/978-3-642-19805-2_23.

35 Pierre Clairambault. Bounding Skeletons, Locally Scoped Terms and Exact Bounds for Linear Head Reduction. In *TLCA*, pages 109–124, 2013. doi:10.1007/978-3-642-38946-7_10.

36 Pierre Crégut. An Abstract Machine for Lambda-Terms Normalization. In *LISP and Functional Programming*, pages 333–340, 1990.

37 Pierre Crégut. Strongly reducing variants of the Krivine abstract machine. *Higher-Order and Symbolic Computation*, 20(3):209–230, 2007. doi:10.1007/s10990-007-9015-z.

38 Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In *ICFP*, pages 233–243, 2000. doi:10.1145/351240.351262.

39 Ugo Dal Lago and Simone Martini. An Invariant Cost Model for the Lambda Calculus. In *CiE 2006*, pages 105–114, 2006. doi:10.1007/11780342_11.

40 Ugo Dal Lago and Simone Martini. Derivational Complexity Is an Invariant Cost Model. In *FOPARA 2009*, pages 100–113, 2009. doi:10.1007/978-3-642-15331-0_7.

41 Ugo Dal Lago and Simone Martini. On Constructor Rewrite Systems and the Lambda-Calculus. In *ICALP (2)*, pages 163–174, 2009. doi:10.1007/978-3-642-02930-1_14.

42 Vincent Danos, Hugo Herbelin, and Laurent Regnier. Game Semantics & Abstract Machines. In *LICS*, pages 394–405, 1996. doi:10.1109/LICS.1996.561456.

43 Vincent Danos and Laurent Regnier. Reversible, Irreversible and Optimal lambda-Machines. *Theor. Comput. Sci.*, 227(1-2):79–97, 1999. doi:10.1016/S0304-3975(99)00049-3.

44 Daniel de Carvalho. Execution time of λ-terms via denotational semantics and intersection types. *Mathematical Structures in Computer Science*, 28(7):1169–1203, 2018. doi:10.1017/S0960129516000396.

45 Daniel de Carvalho, Michele Pagani, and Lorenzo Tortora de Falco. A semantic measure of the execution time in linear logic. *Theor. Comput. Sci.*, 412(20):1884–1902, 2011. doi:10.1016/j.tcs.2010.12.017.

46 Roy Dyckhoff and Stéphane Lengrand. Call-by-Value lambda-calculus and LJQ. *J. Log. Comput.*, 17(6):1109–1134, 2007. doi:10.1093/logcom/exm037.

47 Thomas Ehrhard. Collapsing non-idempotent intersection types. In *CSL*, pages 259–273, 2012. doi:10.4230/LIPIcs.CSL.2012.259.

48 Dan R. Ghica. Slot games: a quantitative model of computation. In *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2005, Long Beach, California, USA, January 12-14, 2005*, pages 85–97, 2005. doi:10.1145/1040305.1040313.

49 Jean-Yves Girard. Linear Logic. *Theoretical Computer Science*, 50:1–102, 1987. doi:10.1016/0304-3975(87)90045-4.

50 Benjamin Grégoire and Xavier Leroy. A compiled implementation of strong reduction. In *(ICFP '02).*, pages 235–246, 2002. doi:10.1145/581478.581501.

51 Hugo Herbelin and Stéphane Zimmermann. An Operational Account of Call-by-Value Minimal and Classical lambda-Calculus in "Natural Deduction" Form. In *TLCA*, pages 142–156, 2009. doi:10.1007/978-3-642-02273-9_12.

52 Andrew Kennedy. Compiling with continuations, continued. In *Proceedings of the 12th ACM SIGPLAN International Conference on Functional Programming, ICFP 2007, Freiburg, Germany, October 1-3, 2007*, pages 177–190, 2007. doi:10.1145/1291151.1291179.

53 Delia Kesner. Reasoning About Call-by-need by Means of Types. In *FOSSACS 2016*, pages 424–441, 2016. doi:10.1007/978-3-662-49630-5_25.

54 Jean-Louis Krivine. *Lambda-calculus, types and models.* Ellis Horwood series in computers and their applications. Masson, 1993.

55 Arne Kutzner and Manfred Schmidt-Schauß. A Non-Deterministic Call-by-Need Lambda Calculus. In *ICFP 1998*, pages 324–335, 1998. doi:10.1145/289423.289462.

56 Ugo Dal Lago, Claudia Faggian, Ichiro Hasuo, and Akira Yoshimizu. The geometry of synchronization. In *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 35:1–35:10, 2014. doi:10.1145/2603088.2603154.

57 Ugo Dal Lago, Claudia Faggian, Benoît Valiron, and Akira Yoshimizu. Parallelism and Synchronization in an Infinitary Context. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 559–572, 2015. doi:10.1109/LICS.2015.58.

58 Ugo Dal Lago, Claudia Faggian, Benoît Valiron, and Akira Yoshimizu. The geometry of parallelism: classical, probabilistic, and quantum effects. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, pages 833–845, 2017. URL: http://dl.acm.org/citation.cfm?id=3009859.

59 Ugo Dal Lago and Martin Hofmann. Quantitative Models and Implicit Complexity. In *FSTTCS 2005: Foundations of Software Technology and Theoretical Computer Science, 25th International Conference, Hyderabad, India, December 15-18, 2005, Proceedings*, pages 189–200, 2005. doi:10.1007/11590156_15.

60 Ugo Dal Lago and Olivier Laurent. Quantitative Game Semantics for Linear Logic. In *Computer Science Logic, 22nd International Workshop, CSL 2008, 17th Annual Conference of the EACSL, Bertinoro, Italy, September 16-19, 2008. Proceedings*, pages 230–245, 2008. doi:10.1007/978-3-540-87531-4_18.

61 Ugo Dal Lago and Simone Martini. On Constructor Rewrite Systems and the Lambda Calculus. *Logical Methods in Computer Science*, 8(3), 2012. doi:10.2168/LMCS-8(3:12)2012.

62 Ugo Dal Lago, Ryo Tanaka, and Akira Yoshimizu. The geometry of concurrent interaction: Handling multiple ports by way of multiple tokens. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005112.

63 Jim Laird, Giulio Manzonetto, Guy McCusker, and Michele Pagani. Weighted Relational Models of Typed Lambda-Calculi. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 301–310, 2013. doi:10.1109/LICS.2013.36.

64 John Lamping. An Algorithm for Optimal Lambda Calculus Reduction. In *POPL*, pages 16–30, 1990. doi:10.1145/96709.96711.

65 John Launchbury. A Natural Semantics for Lazy Evaluation. In *POPL*, pages 144–154, 1993. doi:10.1145/158511.158618.

66 Olivier Laurent and Lorenzo Tortora de Falco. Obsessional Cliques: A Semantic Characterization of Bounded Time Complexity. In *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*, pages 179–188, 2006. doi:10.1109/LICS.2006.37.

67 Jean-Jacques Lévy. Réductions correctes et optimales dans le lambda-calcul. Thése d'Etat, Univ. Paris VII, France, 1978.

68 Jean-Jacques Lévy and Luc Maranget. Explicit Substitutions and Programming Languages. In *Foundations of Software Technology and Theoretical Computer Science, 19th Conference, Chennai, India, December 13-15, 1999, Proceedings*, pages 181–200, 1999. doi:10.1007/3-540-46691-6_14.

69 Ian Mackie. The Geometry of Interaction Machine. In *Conference Record of POPL'95: 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Francisco, California, USA, January 23-25, 1995*, pages 198–208, 1995. doi:10.1145/199448.199483.

70 John Maraist, Martin Odersky, David N. Turner, and Philip Wadler. Call-by-name, Call-by-value, Call-by-need and the Linear lambda Calculus. *Theor. Comput. Sci.*, 228(1-2):175–210, 1999. doi:10.1016/S0304-3975(98)00358-2.

71 John Maraist, Martin Odersky, and Philip Wadler. The Call-by-Need Lambda Calculus. *J. Funct. Program.*, 8(3):275–317, 1998. URL: http://journals.cambridge.org/action/displayAbstract?aid=44169.

72 Damiano Mazza. Simple Parsimonious Types and Logarithmic Space. In Stephan Kreutzer, editor, *24th EACSL Annual Conference on Computer Science Logic (CSL 2015)*, volume 41 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 24–40, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CSL.2015.24.

73 Robin Milner. Functions as Processes. *Math. Str. in Comput. Sci.*, 2(2):119–141, 1992. doi:10.1017/S0960129500001407.

74 Robin Milner. Local Bigraphs and Confluence: Two Conjectures. *Electr. Notes Theor. Comput. Sci.*, 175(3):65–73, 2007. doi:10.1016/j.entcs.2006.07.035.

75 Robin Milner, Joachim Parrow, and David Walker. A Calculus of Mobile Processes, I. *Inf. Comput.*, 100(1):1–40, 1992. doi:10.1016/0890-5401(92)90008-4.

76 Eugenio Moggi. Computational λ-Calculus and Monads. In *LICS '89*, pages 14–23, 1989.

77 Andrzej S. Murawski and C.-H. Luke Ong. Discreet Games, Light Affine Logic and PTIME Computation. In *Computer Science Logic, 14th Annual Conference of the EACSL, Fischbachau, Germany, August 21-26, 2000, Proceedings*, pages 427–441, 2000. doi:10.1007/3-540-44622-2_29.

78 Koko Muroya and Dan R. Ghica. The Dynamic Geometry of Interaction Machine: A Call-by-Need Graph Rewriter. In *CSL 2017*, pages 32:1–32:15, 2017. doi:10.4230/LIPIcs.CSL.2017.32.

79 Luca Paolini. Call-by-Value Separability and Computability. In *ICTCS*, pages 74–89, 2001. doi:10.1007/3-540-45446-2_5.

80 Luca Paolini and Simona Ronchi Della Rocca. Call-by-value Solvability. *ITA*, 33(6):507–534, 1999. doi:10.1051/ita:1999130.

81 Gordon D. Plotkin. Call-by-Name, Call-by-Value and the lambda-Calculus. *Theor. Comput. Sci.*, 1(2):125–159, 1975. doi:10.1016/0304-3975(75)90017-1.

82 Simona Ronchi Della Rocca and Luca Paolini. *The Parametric λ-Calculus*. Springer Berlin Heidelberg, 2004.

83 Amr Sabry and Matthias Felleisen. Reasoning about Programs in Continuation-Passing Style. *Lisp and Symbolic Computation*, 6(3-4):289–360, 1993. doi:10.1007/BF01019462.

84 Amr Sabry and Philip Wadler. A Reflection on Call-by-Value. *ACM Trans. Program. Lang. Syst.*, 19(6):916–941, 1997. doi:10.1145/267959.269968.

85 David Sands, Jörgen Gustavsson, and Andrew Moran. Lambda Calculi and Linear Speedups. In *The Essence of Computation*, pages 60–84, 2002. doi:10.1007/3-540-36377-7_4.

86 Ulrich Schöpp. Space-Efficient Computation by Interaction. In *CSL 2006*, pages 606–621, 2006. doi:10.1007/11874683_40.

87 Peter Sestoft. Deriving a Lazy Abstract Machine. *J. Funct. Program.*, 7(3):231–264, 1997. URL: http://journals.cambridge.org/action/displayAbstract?aid=44087.

88 Cees F. Slot and Peter van Emde Boas. On Tape Versus Core; An Application of Space Efficient Perfect Hash Functions to the Invariance of Space. In *STOC 1984*, pages 391–400, 1984. doi:10.1145/800057.808705.

89 Christopher P. Wadsworth. *Semantics and pragmatics of the lambda-calculus*. PhD Thesis, Oxford, 1971.

A Linear Logical Framework in Hybrid

Amy P. Felty
University of Ottawa, Canada
http://www.site.uottawa.ca/~afelty/
afelty@uottawa.ca

Abstract

We present a linear logical framework implemented within the Hybrid system [2]. Hybrid is designed to support the use of higher-order abstract syntax for representing and reasoning about formal systems, implemented in the Coq Proof Assistant. In this work, we extend the system with two linear specification logics, which provide infrastructure for reasoning directly about object languages with linear features.

We originally developed this framework in order to address the challenges of reasoning about the type system of a quantum lambda calculus. In particular, we started by considering the Proto-Quipper language [6], which contains the core of Quipper [3, 7]. Quipper is a relatively new quantum programming language under active development with a linear type system. We have completed a formal proof of type soundness for Proto-Quipper [5]. Our current work includes extending this work to other properties of Proto-Quipper, reasoning about other quantum programming languages [4], and reasoning about other languages such as the meta-theory of low-level abstract machine code.

We are also interested in applying this framework to applications outside the domain of meta-theory of programming languages and have focused on two areas – formal reasoning about the proof theory of focused linear sequent calculi and modeling biological processes as transition systems and proving properties about them. We found that a slight extension of the initial linear specification logic allowed us to provide succinct encodings and facilitate reasoning in these new domains. We illustrate by discussing a model of breast cancer progression as a set of transition rules and proving properties about this model [1]. Current work also includes modeling stem cells as they mature into different types of blood cells.

This work illustrates the use of Hybrid as a meta-logical framework for fast prototyping of logical frameworks, which is achieved by defining inference rules of a specification logic inductively in Coq and building a library of definitions and lemmas used to reason about a class of object logics. Our focus here is on linear specification logics and their applications.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Operational semantics; Theory of computation → Type theory; Theory of computation → Functional constructs; Theory of computation → Type structures

Keywords and phrases Logical frameworks, proof assistants, linear logic

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.2

Category Invited Talk

Funding *Amy P. Felty*: The author acknowledges the support of the Natural Sciences and Engineering Research Council of Canada.

References

1 Joëlle Despeyroux, Amy Felty, Pietro Lio, and Carlos Olarte. A Logical Framework for Modelling Breast Cancer Progression. In *International Symposium on Molecular Logic and Computational Synthetic Biology (MLCSB)*, 2018.

2 Amy P. Felty and Alberto Momigliano. Hybrid: A Definitional Two-Level Approach to Reasoning with Higher-Order Abstract Syntax. *Journal of Automated Reasoning*, 48(1):43–105, 2012.

© Amy P. Felty;
licensed under Creative Commons License CC-BY
4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019).
Editor: Herman Geuvers; Article No. 2; pp. 2:1–2:2
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

3 Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. Quipper: A Scalable Quantum Programming Language. In *Thirty-Fourthh ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 333–342. ACM, 2013.

4 Mohamed Yousri Mahmoud and Amy P. Felty. Formal Meta-level Analysis Framework for Quantum Programming Languages. In *12th Workshop on Logical and Semantic Frameworks with Applications (LSFA 2017)*, volume 338 of *Electronic Notes in Theoretical Computer Science*, pages 185–201, 2018.

5 Mohamed Yousri Mahmoud and Amy P. Felty. Formalization of Metatheory of the Quipper Quantum Programming Language in a Linear Logic. *CoRR*, 2018. `arXiv:1812.03624`.

6 Neil J. Ross. *Algebraic and Logical Methods in Quantum Computation.* PhD thesis, Dalhousie University, August 2015. `arXiv:1510.02198`.

7 Peter Selinger and Benoît Valiron. A Lambda Calculus for Quantum Computation with Classial Control. *Mathematical Structures in Computer Science*, 16(3):527–552, 2006.

Extending Maximal Completion

Sarah Winkler
University of Innsbruck, Austria
http://cl-informatik.uibk.ac.at/users/swinkler
sarah.winkler@uibk.ac.at

Abstract

Maximal completion (Klein and Hirokawa 2011) is an elegantly simple yet powerful variant of Knuth-Bendix completion. This paper extends the approach to ordered completion and theorem proving as well as normalized completion. An implementation of the different procedures is described, and its practicality is demonstrated by various examples.

2012 ACM Subject Classification Theory of computation → Automated reasoning

Keywords and phrases automated reasoning, completion, theorem proving

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.3

Category Invited Talk

Funding Supported by FWF (Austrian Science Fund) project T789.

Acknowledgements The author is grateful to Deepak Kapur for his insightful comments.

1 Introduction

Knuth-Bendix completion [18] constitutes a milestone in the history of equational theorem proving and automated deduction in general. Given a set of input equalities $\mathcal{E}_0$, it can generate a presentation of the equational theory as a complete rewrite system $\mathcal{R}$ which may serve to decide the validity problem for the theory.

▶ **Example 1.** In order to simplify proofs found by SMT solvers, Wehrman and Stump [32] pursue an algebraic approach: proofs are represented by first-order terms, and the equivalences usable for simplification are described by 20 equations like the following ones:

$$\begin{array}{ccc}
(x \cdot y) \cdot z \approx x \cdot (y \cdot z) & (\mathsf{refl} \cdot x) \approx x & (x \cdot \mathsf{refl}) \approx x \\
\mathsf{or}_1(\mathsf{refl}) \approx \mathsf{refl} & \mathsf{and}_1(\mathsf{refl}) \approx \mathsf{refl} & \mathsf{not}(\mathsf{refl}) \approx \mathsf{refl} \\
\mathsf{or}_1(x) \cdot \mathsf{or}_2^\mathsf{T} \approx \mathsf{or}_2^\mathsf{T} & \mathsf{and}_1(x) \cdot \mathsf{and}_2^\mathsf{F} \approx \mathsf{and}_2^\mathsf{F} & \mathsf{or}_2(x) \cdot \mathsf{or}_1^\mathsf{F} \approx (\mathsf{or}_1^\mathsf{F} \cdot x) \\
\mathsf{or}_1(x) \cdot \mathsf{or}_2^\mathsf{F} \approx (\mathsf{or}_2^\mathsf{F} \cdot x) & \mathsf{not}(x) \cdot \mathsf{not}(y) \approx \mathsf{not}(x \cdot y) & \mathsf{or}_2(x) \cdot \mathsf{or}_1^\mathsf{T} \approx \mathsf{or}_1^\mathsf{T}
\end{array}$$

Here $\cdot$ denotes concatenation, refl is the reflexivity proof, the symbols and_i, or_i and not are used for congruence, and constants like or_1^T stand for operations with boolean constants.

A Knuth-Bendix completion procedure can transform this set of equations into a terminating and confluent rewrite system $\mathcal{R}$ consisting of 45 rules, including the following:

$$\begin{array}{ccc}
(x \cdot y) \cdot z \to x \cdot (y \cdot z) & \mathsf{or}_1(x) \cdot \mathsf{or}_2^\mathsf{T} \to \mathsf{or}_2^\mathsf{T} & (\mathsf{refl} \cdot x) \to x \\
(\mathsf{or}_2(x) \cdot \mathsf{or}_1(y)) \cdot \mathsf{or}_1^\mathsf{T} \to \mathsf{or}_1(y) \cdot \mathsf{or}_1^\mathsf{T} & \mathsf{or}_2(x) \cdot \mathsf{or}_1^\mathsf{T} \to \mathsf{or}_1^\mathsf{T} & \mathsf{or}_1(\mathsf{refl}) \to \mathsf{refl} \\
(x \cdot \mathsf{and}_2(y)) \cdot \mathsf{and}_2(z) \to x \cdot \mathsf{and}_2(y \cdot z) & \mathsf{or}_2(\mathsf{refl}) \to \mathsf{refl} & \mathsf{or}_2(x) \cdot \mathsf{or}_1^\mathsf{T} \to \mathsf{or}_1^\mathsf{T}
\end{array}$$

This rewrite system can be used to simplify an arbitrary proof (represented by a term) into its unique normal form. Moreover, any two proofs can be tested for equivalence simply by checking whether their normal forms are the same.

© Sarah Winkler;
licensed under Creative Commons License CC-BY
4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019).
Editor: Herman Geuvers; Article No. 3; pp. 3:1–3:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

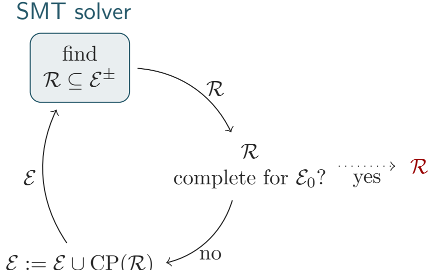


Figure 1 Maximal completion.

Knuth and Bendix presented completion as a concrete algorithm. Pioneered by Bachmair, Dershowitz, and Hsiang [5], it is nowadays more common to describe completion by an inference system, thus abstracting from concrete implementations.

More recently, Klein and Hirokawa [17] proposed a radically different approach: *Maximal completion* first approximates a complete presentation by extracting a terminating rewrite system from an equation pool. It then checks whether the candidate system is complete, and if a counterexample was found the procedure is repeated with an extended equation pool. Figure 1 illustrates the approach. Maximal completion has the advantage that the reduction order, a typically critical input parameter, need not be fixed in advance and can be changed at any point. The candidate rewrite systems are generated by means of SAT/SMT solvers; thus also advanced termination methods can be used in this setting and the search can be guided towards different objectives [25]. Despite the simple, declarative formulation of the procedure, the authors' implementation resulted in a competitive tool [17, 25].

Apart from these improvements of classical Knuth-Bendix completion, numerous variants have by now joined the family of completion calculi, aiming to make completion more versatile and powerful. One of the most prominent variants is ordered completion. It was developed by Bachmair, Dershowitz, and Plaisted to remedy the shortcoming that classical completion fails if unorientable equations like commutativity are encountered [6].

Another line of research tackled the development of dedicated completion procedures for equational systems which incorporate common algebraic theories such as associativity and commutativity [23, 12]. The latest and most generally applicable method of this kind is normalized completion, developed by Marché [22].

In this paper maximal completion is revisited (Section 3) and extended to ordered and normalized completion. More specifically, the contributions of this paper are as follows:

- Maximal ordered completion and an according equational theorem proving method are explained in detail. In particular a completeness proof is presented, showing that a ground complete system can always be found.
- The proofs for (ordered) completion require only *prime* critical pairs to be considered.
- For the case of linear input equalities, it is proven that even a complete system can be found if it exists (Section 4.2), and a bound on the number of iterations is derived.
- A maximal completion version of normalized completion (Section 5) is presented. This covers AC completion, as well as the computation of Gröbner bases [22].

Section 6 is devoted to the implementation of these procedures in the tool MædMax. Some example use cases from different application areas are demonstrated along the way. Finally, Section 7 concludes.

2 Preliminaries

In the sequel familiarity with the basics of term rewriting is assumed [2], but some key notions are recalled in this section. Let $\mathcal{T}(\mathcal{F},\mathcal{V})$ denote the set of all terms over a signature $\mathcal{F}$ and an infinite set of variables $\mathcal{V}$, and $\mathcal{T}(\mathcal{F})$ the set of all *ground* terms over $\mathcal{F}$. A *substitution* σ is a mapping from variables to terms. As usual, $t\sigma$ denotes the application of σ to the term t. A pair of terms (s,t) is sometimes considered an *equation*, which is expressed by writing $s \approx t$, and sometimes a *(rewrite) rule*, denoted $s \to t$. An equational system (ES) is a set of equations, a term rewrite system (TRS) is a set of rewrite rules. Given an ES $\mathcal{E}$, we write $\mathcal{E}^{\pm}$ to denote its *symmetric closure* $\mathcal{E} \cup \{t \approx s \mid s \approx t \in \mathcal{E}\}$. A *reduction order* is a proper and well-founded order on terms which is closed under contexts and substitutions. It is *ground total* if it is total on $\mathcal{T}(\mathcal{F})$. In the remainder most examples use the Knuth-Bendix order (KBO), written $>_{\mathsf{kbo}}$, and the lexicographic path order (LPO), written $>_{\mathsf{lpo}}$.

A TRS $\mathcal{R}$ is *terminating* if $\to_{\mathcal{R}}$ is well-founded. It is *(ground) confluent* if $s \mathrel{{}_{\mathcal{R}}^{*}{\leftarrow}} \cdot \to_{\mathcal{R}}^{*} t$ implies $s \to_{\mathcal{R}}^{*} \cdot \mathrel{{}_{\mathcal{R}}^{*}{\leftarrow}} t$ for all (ground) terms s and t. It is *(ground) complete* if it is terminating and (ground) confluent. We say that $\mathcal{R}$ is a *complete presentation* of an ES $\mathcal{E}$ if $\mathcal{R}$ is complete and $\leftrightarrow_{\mathcal{R}}^{*} = \leftrightarrow_{\mathcal{E}}^{*}$. Similarly, $\mathcal{R}$ is a *ground complete presentation* of an ES $\mathcal{E}$ if $\mathcal{R}$ is ground complete and the equivalence $\leftrightarrow_{\mathcal{R}}^{*} = \leftrightarrow_{\mathcal{E}}^{*}$ holds on ground terms. For a TRS $\mathcal{R}$ and terms s and t, the notation $s \downarrow_{\mathcal{R}} t$ expresses existence of a joining sequence $s \to_{\mathcal{R}}^{*} \cdot \mathrel{{}_{\mathcal{R}}^{*}{\leftarrow}} t$. If $\mathcal{R}$ is terminating then $t{\downarrow_{\mathcal{R}}}$ denotes some fixed normal form of t, and $\mathsf{NF}(\mathcal{R})$ denotes the set of all normal forms of $\mathcal{R}$. This notation is extended to ESs $\mathcal{E}$ by writing $\mathcal{E}{\downarrow_{\mathcal{R}}}$ for the ES $\{s{\downarrow_{\mathcal{R}}} \approx t{\downarrow_{\mathcal{R}}} \mid s \approx t \in \mathcal{E} \text{ and } s{\downarrow_{\mathcal{R}}} \neq t{\downarrow_{\mathcal{R}}}\}$.

Completion procedures are based on critical pair analysis. To that end, an *overlap* of a TRS $\mathcal{R}$ is a triple $\langle \ell_1 \to r_1, p, \ell_2 \to r_2 \rangle$ such that $\ell_1 \to r_1$ and $\ell_2 \to r_2$ are variants of rules in $\mathcal{R}$ without common variables, $p \in \mathcal{P}\mathrm{os}_{\mathcal{F}}(\ell_2)$, ℓ_1 and $\ell_2|_p$ are unifiable, and if $p = \epsilon$ then $\ell_1 \to r_1$ and $\ell_2 \to r_2$ are not variants of each other. Suppose $\langle \ell_1 \to r_1, p, \ell_2 \to r_2 \rangle$ is an overlap of a TRS $\mathcal{R}$ and σ is a most general unifier of ℓ_1 and $\ell_2|_p$. Then the equation $\ell_2[r_1]_p\sigma \approx r_2\sigma$ is a *critical pair* of $\mathcal{R}$. The set of all critical pairs of $\mathcal{R}$ is denoted by $\mathrm{CP}(\mathcal{R})$. A critical pair is *prime* if no proper subterm of $\ell_1\sigma$ is reducible in $\mathcal{R}$. The set of all prime critical pairs of $\mathcal{R}$ is denoted by $\mathrm{PCP}(\mathcal{R})$. It is known that only prime critical pairs need to be considered for confluence of terminating TRSs:

▶ **Lemma 2** ([14]). *A terminating TRS $\mathcal{R}$ is confluent if and only if* $\mathrm{PCP}(\mathcal{R}) \subseteq {\downarrow_{\mathcal{R}}}$.

Further preliminaries will be introduced in later sections as necessary.

3 Maximal Completion

This section recapitulates the maximal completion approach by Klein and Hirokawa [17]. A TRS $\mathcal{R}$ is said to be *over* an ES $\mathcal{E}$ if $\mathcal{R} \subseteq \mathcal{E}^{\pm}$. The set of all terminating TRSs $\mathcal{R}$ over $\mathcal{E}$ is denoted $\mathfrak{T}(\mathcal{E})$. We assume two functions $\mathfrak{R}$ and Ext such that $\mathfrak{R}(\mathcal{E}) \subseteq \mathfrak{T}(\mathcal{E})$ returns a set of terminating TRSs over $\mathcal{E}$, and the extension function Ext satisfies $\mathrm{Ext}(\mathcal{E}) \subseteq \leftrightarrow_{\mathcal{E}}^{*}$ for all ESs $\mathcal{E}$. We define maximal completion by means of the following transformation.

▶ **Definition 3.** *Given a set of input equalities $\mathcal{E}_0$ and an ES $\mathcal{E}$, let*

$$\varphi(\mathcal{E}) = \begin{cases} \mathcal{R} & \text{if } \mathcal{R} \in \mathfrak{R}(\mathcal{E}) \text{ such that } \mathrm{PCP}(\mathcal{R}) \cup \mathcal{E}_0 \subseteq {\downarrow_{\mathcal{R}}} \\ \varphi(\mathcal{E} \cup \mathrm{Ext}(\mathcal{E})) & \text{otherwise.} \end{cases}$$

Note that this definition differs from [17, Definition 2] by the use of prime critical pairs. In general φ does not need to be defined, nor is it necessarily unique. But if $\varphi(\mathcal{E}_0)$ is defined then we can assume a sequence of ESs $\mathcal{E}_1, \ldots, \mathcal{E}_k$ called *maximal completion sequence*

such that $\mathcal{E}_{i+1} = \mathcal{E}_i \cup \mathrm{Ext}(\mathcal{E}_i)$ for all $0 \leqslant i < k$, and there is some $\mathcal{R} \in \mathfrak{R}(\mathcal{E}_k)$ such that $\mathrm{PCP}(\mathcal{R}) \cup \mathcal{E}_0 \subseteq {\downarrow_\mathcal{R}}$. The following theorem expresses correctness of maximal completion [17, Theorem 3]:

▶ **Lemma 4.** *If $\varphi(\mathcal{E}_0)$ is defined then it is a complete presentation of $\mathcal{E}_0$.*

Proof. Let $\varphi(\mathcal{E}_0) = \mathcal{R}$ and $\mathcal{E}_1, \ldots, \mathcal{E}_k$ be an according maximal completion sequence. The TRS $\mathcal{R}$ must be terminating since it was returned by $\mathfrak{R}$. Because of $\mathrm{PCP}(\mathcal{R}) \subseteq {\downarrow_\mathcal{R}}$ it is confluent by Lemma 2, and hence complete.

A simple induction argument using the global assumption $\mathrm{Ext}(\mathcal{E}) \subseteq {\leftrightarrow^*_\mathcal{E}}$ for all ESs $\mathcal{E}$ shows that $\mathcal{E}_i \subseteq {\leftrightarrow^*_{\mathcal{E}_0}}$ for all $i \geqslant 0$. Since $\mathcal{R}$ is over $\mathcal{E}_k$, also ${\leftrightarrow^*_\mathcal{R}} \subseteq {\leftrightarrow^*_{\mathcal{E}_0}}$ holds. Conversely, $\mathcal{E}_0 \subseteq {\downarrow_\mathcal{R}}$ ensures ${\leftrightarrow^*_{\mathcal{E}_0}} \subseteq {\leftrightarrow^*_\mathcal{R}}$. So $\mathcal{R}$ is a complete presentation of $\mathcal{E}_0$. ◀

Note that maximal completion is based on just three ingredients: (1) completeness is overapproximated by termination using the function $\mathfrak{R}$, (2) a success check determines whether some TRS $\mathcal{R} \in \mathfrak{R}(\mathcal{E})$ is complete, and (3) the current set of equations $\mathcal{E}$ is extended by means of a theory-preserving function Ext.

It is natural to choose $\mathrm{Ext}(\mathcal{E})$ such that $\mathrm{Ext}(\mathcal{E}) \subseteq \bigcup_{\mathcal{R} \in \mathfrak{R}(\mathcal{E})} \mathrm{CP}(\mathcal{R}){\downarrow_\mathcal{R}}$. Klein and Hirokawa moreover proposed $\mathfrak{R}(\mathcal{E})$ to return elements of $\mathfrak{T}(\mathcal{E})$ with maximal cardinality, hence the name. The rationale for this choice is that adding rules to a complete presentation $\mathcal{R}$ of $\mathcal{E}_0$ does not hurt this property, as long as termination and the equational theory are preserved. This is formally expressed by the following lemma.

▶ **Lemma 5** ([17, Lemma 4])**.** *Let $\mathcal{R}$ be a complete presentation of $\mathcal{E}_0$ and $\mathcal{R}'$ a terminating TRS such that $\mathcal{R} \subseteq \mathcal{R}' \subseteq {\leftrightarrow^*_{\mathcal{E}_0}}$. Then also $\mathcal{R}'$ is a complete presentation of $\mathcal{E}_0$.* ◀

Nevertheless a maximal terminating TRS may constitute an unfortunate choice in maximal completion, as illustrated by the next example.

▶ **Example 6.** Let $\mathcal{E}_0$ consist of the following four equations:

$$x + 0 \approx x \qquad \mathsf{s}(x+y) \approx x + \mathsf{s}(y) \qquad \mathsf{z}(x) \approx 0 \qquad \mathsf{z}(\mathsf{s}(x+y)) \approx \mathsf{z}(x + \mathsf{s}(0))$$

Let $\mathcal{R}_1$ be the TRS obtained by orienting all equations from left to right:

$$x + 0 \to x \qquad \mathsf{s}(x+y) \to x + \mathsf{s}(y) \qquad \mathsf{z}(x) \to 0 \qquad \mathsf{z}(\mathsf{s}(x+y)) \to \mathsf{z}(x + \mathsf{s}(0))$$

Termination of $\mathcal{R}_1$ can e.g. be verified using a KBO with $\mathsf{s} > +$ and $w_0 = w(f) = 1$ for all function symbols f. Thus $\mathfrak{R}(\mathcal{E}_0) = \{\mathcal{R}_1\}$ is a valid choice for maximal completion. Now the first two rules admit the overlap $\mathsf{s}(x) \leftarrow \mathsf{s}(x+0) \to x + \mathsf{s}(0)$ which creates an irreducible critical pair $\mathsf{s}(x) \approx x + \mathsf{s}(0)$. There are also three critical pairs involving the last rule, but they are all joinable. Let thus $\mathcal{E}_1$ be $\mathcal{E}_0 \cup \{\mathsf{s}(x) \approx x + \mathsf{s}(0)\}$. Using the same reduction order, all equations can be oriented into the TRS $\mathcal{R}_2 = \mathcal{R}_1 \cup \{x + \mathsf{s}(0) \to \mathsf{s}(x)\}$. Suppose $\mathfrak{R}(\mathcal{E}_1)$ is $\{\mathcal{R}_2\}$. There is only one new non-joinable overlap: $\mathsf{s}(\mathsf{s}(x)) \leftarrow \mathsf{s}(x + \mathsf{s}(0)) \to x + \mathsf{s}(\mathsf{s}(0))$, so let $\mathcal{E}_2 = \mathcal{E}_1 \cup \{\mathsf{s}(\mathsf{s}(x)) \approx x + \mathsf{s}(\mathsf{s}(0))\}$. Repeating this strategy will fail to produce a finite complete system, as it gives rise to infinitely many equations $\mathsf{s}^n(x) \approx x + \mathsf{s}^n(0)$.

So this reduction order does not lead to a finite complete presentation of $\mathcal{E}_0$. But in fact $\mathcal{R}_1$ is the only terminating TRS over $\mathcal{E}_0$ which has four rules: This is because the last equation can only be oriented from left to right, and the second cannot be oriented from right to left in combination with the last without violating termination.

Suppose that $\mathfrak{R}(\mathcal{E}_0)$ contains instead the following TRS $\mathcal{R}'_1$ which has only three rules:

$$x + 0 \to x \qquad x + \mathsf{s}(y) \to \mathsf{s}(x+y) \qquad \mathsf{z}(x) \to 0$$

Termination of $\mathcal{R}'_1$ can be shown by changing the precedence in the above KBO to $+ > \mathsf{s}$. There are no critical pairs, and $\mathcal{R}'_1$ joins the input equalities $\mathcal{E}_0$. So maximal completion can succeed immediately by returning $\mathcal{R}'_1$.

In the implementation in the tool MædMax the function $\mathfrak{R}$ chooses rewrite systems $\mathcal{R}$ over $\mathcal{E}$ which can *reduce* rather than *orient* a maximal number of equations in $\mathcal{E}$. Note that the TRS $\mathcal{R}_1'$ in Example 6 is optimal in this sense, since it reduces all equations in $\mathcal{E}_0$.

▶ **Example 7.** In nine iterations of maximal completion, that is within nine recursive calls of the procedure φ, the proof reduction system described in Example 1 can be transformed into a complete rewrite system $\mathcal{R}$. The maximal completion run produces 150 equations and takes about 10 seconds. It is worth noting that to complete this system, LPO or KBO alone do not suffice; advanced termination techniques like dependency pairs are required, see [25].

4 Ordered Completion and Theorem Proving

This section is devoted to the extension of maximal completion to ordered completion and equational theorem proving. The basic procedure was already outlined in [36].

First some concepts specific to this setting are introduced. In this section a ground total reduction order $>$ is considered, unless stated otherwise. Given a reduction order $>$ and an ES $\mathcal{E}$, the *ordered rewrite system* $\mathcal{E}_>$ consists of all rules $s\sigma \to t\sigma$ such that $s \approx t \in \mathcal{E}$ and $s\sigma > t\sigma$. A triple $(\mathcal{R}, \mathcal{E}, >)$ of a TRS $\mathcal{R}$, an ES $\mathcal{E}$, and a reduction order $>$ is called *ground complete* if the (possibly infinite) TRS $\mathcal{R} \cup \mathcal{E}_>$ is. An equation $s \approx t$ is *ground joinable* over a TRS $\mathcal{R}$ if $s\sigma \downarrow_{\mathcal{R}} t\sigma$ for all grounding substitutions σ. Ordered completion uses a relaxed definition of critical pairs. Given a reduction order $>$ and an ES $\mathcal{E}$, an *extended overlap* consists of two variable-disjoint variants $\ell_1 \approx r_1$ and $\ell_2 \approx r_2$ of equations in $\mathcal{E}^{\pm}$ such that $p \in \mathcal{P}\mathrm{os}_{\mathcal{F}}(\ell_2)$ and ℓ_1 and $\ell_2|_p$ are unifiable with most general unifier σ. An extended overlap which satisfies $r_1\sigma \not> \ell_1\sigma$ and $r_2\sigma \not> \ell_2\sigma$ gives rise to the *extended critical pair* $\ell_2[r_1]_p\sigma \approx r_2\sigma$. The set $\mathrm{CP}_>(\mathcal{E})$ consists of all extended critical pairs between equations in $\mathcal{E}$. An extended critical pair is *prime* if all proper subterms of $\ell_1\sigma$ are $\mathcal{E}_>$-normal forms. The set of prime extended critical pairs among equations in $\mathcal{E}$ is denoted by $\mathrm{PCP}_>(\mathcal{E})$.

Next, an ordered version of maximal completion gets defined. Let $\mathfrak{R}_{\mathsf{o}}$ be a function such that $\mathfrak{R}_{\mathsf{o}}(\mathcal{E}) \subseteq \mathfrak{T}(\mathcal{E})$ returns a set of *totally terminating* TRSs over $\mathcal{E}$, that is TRSs $\mathcal{R}$ which are contained in a ground total reduction order $>$. Moreover, the extension function $\mathrm{Ext}_{\mathsf{o}}$ is supposed to satisfy $\mathrm{Ext}_{\mathsf{o}}(\mathcal{E}) \subseteq \leftrightarrow_{\mathcal{E}}^*$ for all ESs $\mathcal{E}$.

▶ **Definition 8.** *Given a set of input equalities $\mathcal{E}_0$ and an ES $\mathcal{E}$, let*

$$\varphi_{\mathsf{o}}(\mathcal{E}) = \begin{cases} (\mathcal{R}, \mathcal{E}{\downarrow_{\mathcal{R}}}, >) & \text{if } \mathcal{R} \in \mathfrak{R}_{\mathsf{o}}(\mathcal{E}) \text{ and all equations in } \mathcal{E}_0 \cup \mathrm{PCP}_>(\mathcal{E}{\downarrow_{\mathcal{R}}} \cup \mathcal{R}) \\ & \text{are ground joinable in } \mathcal{R} \cup (\mathcal{E}{\downarrow_{\mathcal{R}}})_> \\ \varphi_{\mathsf{o}}(\mathcal{E} \cup \mathrm{Ext}_{\mathsf{o}}(\mathcal{E})) & \text{otherwise.} \end{cases}$$

In order to show correctness of this procedure, the following auxiliary result is useful:

▶ **Lemma 9.** *Suppose $\mathcal{R} \subseteq >$, $\mathcal{R} \cup \mathcal{E} \subseteq \leftrightarrow_{\mathcal{E}_0}^*$ and all equations in $\mathcal{E}_0 \cup \mathrm{PCP}_>(\mathcal{E} \cup \mathcal{R})$ are ground joinable in $\mathcal{R} \cup \mathcal{E}_>$. Then $(\mathcal{R}, \mathcal{E}, >)$ is a ground complete presentation of $\mathcal{E}_0$.*

Proof. Let $\mathcal{S}$ denote the TRS $\mathcal{R} \cup \mathcal{E}_>$, which terminates because it is contained in $>$. We can thus show ground confluence of $\mathcal{S}$ via local ground confluence. The inclusion

$$\xleftrightarrow[\mathrm{PCP}(\mathcal{S})]{} \subseteq \xleftrightarrow[\mathrm{PCP}_>(\mathcal{R} \cup \mathcal{E})]{} \cup \downarrow_{\mathcal{S}} \tag{1}$$

holds on ground terms according to [11, Lemma 26]. By assumption we have $\mathcal{S}$-ground joinability of $\mathrm{PCP}_>(\mathcal{R} \cup \mathcal{E})$, and hence $\leftrightarrow_{\mathrm{PCP}(\mathcal{S})} \subseteq \downarrow_{\mathcal{S}}$ on ground terms. So by Lemma 2 the TRS $\mathcal{S}$ is confluent on ground terms.

Since $\mathcal{R} \cup \mathcal{E} \subseteq \leftrightarrow^*_{\mathcal{E}_0}$ was assumed, also $\leftrightarrow^*_{\mathcal{S}} \subseteq \leftrightarrow^*_{\mathcal{E}_0}$ holds. Moreover $\mathcal{E}_0$ is $\mathcal{S}$-ground joinable by assumption. Hence the equivalence $\leftrightarrow^*_{\mathcal{S}} = \leftrightarrow^*_{\mathcal{E}_0}$ is satisfied on ground terms, so $\mathcal{S}$ is a ground complete presentation of $\mathcal{E}_0$. ◀

Now correctness of the transformation φ_{o} is obvious:

▶ **Lemma 10.** *If $\varphi_{\mathsf{o}}(\mathcal{E}_0)$ is defined then it is a ground complete presentation of $\mathcal{E}_0$.*

Note that Definition 8 uses the idea of Definition 3 in the setting of ground completeness but suffers the major drawback of an undecidable success check since ground joinability of ordered rewriting is undecidable [20]. An implementation thus has to rely on sufficient ground joinability criteria, an example of which is stated next. Its correctness follows from the more sophisticated test presented in [33].

▶ **Lemma 11.** *An equation $s \approx t$ is ground joinable in $\mathcal{R} \cup \mathcal{E}_>$ if $s \downarrow_{\mathcal{R}} t$ or $s{\downarrow_{\mathcal{R}}} \approx t{\downarrow_{\mathcal{R}}} \in \mathcal{E}$.*

In our implementation $\mathrm{Ext}_{\mathsf{o}}(\mathcal{E})$ is chosen as a subset of $\bigcup_{\mathcal{R} \in \mathfrak{R}_{\mathsf{o}}(\mathcal{E})} \mathrm{PCP}_>(\mathcal{R} \cup \mathcal{E}{\downarrow_{\mathcal{R}}}){\downarrow_{\mathcal{R}}}$.

Bachmair, Dershowitz, and Plaisted showed that their ordered completion procedures always succeed in producing a ground complete system (though possibly in the limit) [6]. Next, we derive a similar property for maximal ordered completion, under the assumption that all prime critical pairs are considered. To this end, we consider an infinite maximal ordered completion sequence $\mathcal{E}_0, \mathcal{E}_1, \mathcal{E}_2, \ldots$ such that $\mathcal{E}_{i+1} = \mathcal{E}_i \cup \mathrm{Ext}_{\mathsf{o}}(\mathcal{E}_i)$ for all $i \geqslant 0$. Let moreover $\mathcal{E}^\infty$ denote the limit $\bigcup_i \mathcal{E}_i$. The following statement holds by the global assumption on $\mathrm{Ext}_{\mathsf{o}}$.

▶ **Lemma 12.** *The conversion equivalence $\leftrightarrow^*_{\mathcal{E}_0} = \leftrightarrow^*_{\mathcal{E}_i}$ holds for all $i \geqslant 0$.*

It is known that ground complete systems remain ground complete when they get (moderately) reduced, the following result follows from Lemma 5 and [11, Theorem 43].

▶ **Lemma 13.** *If $\mathcal{R} \cup \mathcal{E}_>$ is ground complete then so is $\mathcal{R} \cup (\mathcal{E}{\downarrow_{\mathcal{R}}})_>$.*

Next we show the main completeness result for maximal ordered completion:

▶ **Theorem 14.** *Suppose $\mathrm{Ext}_{\mathsf{o}}(\mathcal{E}) \supseteq \bigcup_{\mathcal{R} \in \mathfrak{R}_{\mathsf{o}}(\mathcal{E})} \mathrm{PCP}_>(\mathcal{R} \cup \mathcal{E}){\downarrow_{\mathcal{R}}}$ for all ESs $\mathcal{E}$. For any $\mathcal{R} \in \mathfrak{R}_{\mathsf{o}}(\mathcal{E}^\infty)$ the system $\mathcal{R} \cup (\mathcal{E}^\infty{\downarrow_{\mathcal{R}}})_>$ is a ground complete presentation of $\mathcal{E}_0$.*

Proof. Let $\mathcal{R} \in \mathfrak{R}_{\mathsf{o}}(\mathcal{E}^\infty)$. The following arguments show that $\mathcal{S} = \mathcal{R} \cup (\mathcal{E}^\infty)_>$ is ground complete. The claim then follows from Lemma 13.

The TRS $\mathcal{S}$ is terminating because $\mathcal{S} \subseteq >$. In order to show that $\mathcal{S}$ considered as a TRS on ground terms is also confluent, according to Lemma 2 applied to $\mathcal{S}$ it suffices to show that all prime critical pairs of $\mathcal{S}$ are joinable. So consider an equation $s \approx t \in \mathrm{PCP}(\mathcal{S})$. Like in the proof of Lemma 9, we can use [11, Lemma 26] to obtain inclusion (1). So we have $s \downarrow_{\mathcal{S}} t$, or there is some equation $u \approx v \in \mathrm{PCP}_>(\mathcal{R} \cup \mathcal{E}^\infty)$ such that $s \leftrightarrow_{u \approx v} t$. In the former case, there is nothing to show. Otherwise, we have $u{\downarrow_{\mathcal{R}}} \approx v{\downarrow_{\mathcal{R}}} \in \mathrm{Ext}_{\mathsf{o}}(\mathcal{E}^\infty) \subseteq \mathcal{E}^\infty$ by assumption. But then $u{\downarrow_{\mathcal{R}}} \approx v{\downarrow_{\mathcal{R}}}$ is $\mathcal{S}$-ground joinable by Lemma 11, and hence $s \approx t$ is joinable.

By Lemma 12 and the definition of $\mathcal{E}^\infty$, the inclusion $\mathcal{E}^\infty \subseteq \leftrightarrow^*_{\mathcal{E}_0}$ holds. The equivalence $\leftrightarrow^*_{\mathcal{E}^\infty} = \leftrightarrow^*_{\mathcal{E}_0}$ thus follows from $\mathcal{E}_0 \subseteq \mathcal{E}^\infty$. Because $>$ is ground complete, $\leftrightarrow_{\mathcal{S}} = \leftrightarrow_{\mathcal{E}^\infty}$ holds on ground terms, which implies $\leftrightarrow^*_{\mathcal{S}} = \leftrightarrow^*_{\mathcal{E}_0}$. ◀

▶ **Example 15.** Consider the following ES $\mathcal{E}_0$ axiomatizing a Boolean ring, where multiplication is denoted by concatenation.

(1)	$(x+y)+z \approx x+(y+z)$	(2)	$x+y \approx y+x$	(3)	$0+x \approx x$
(4)	$x(y+z) \approx xy+xz$	(5)	$(xy)z \approx x(yz)$	(6)	$xy \approx yx$
(7)	$(x+y)z \approx xz+yz$	(8)	$xx \approx x$	(9)	$x+x \approx 0$
(10)	$1x \approx x$				

Let (i) denote equation (i) oriented from left to right, and $(\bar{\text{i}})$ the reverse orientation. Suppose $\mathcal{R}_1$ is the TRS $\{(1),(3),(\bar{4}),(5),(\bar{7}),(8),(9),(10)\}$, and $\mathfrak{R}_{\mathsf{o}}(\mathcal{E}_0) = \{\mathcal{R}_1\}$. Now the set $\mathrm{Ext}_{\mathsf{o}}(\mathcal{E}_0)$ may consist of the following extended critical pairs of rules among $\mathcal{R}_1$ and the unorientable commutativity equation:

(11)	$x+(y+z) \approx y+(x+z)$	(12)	$x(yz) \approx y(xz)$	(13)	$x+0 \approx x$
(14)	$y+(x+y) \approx x$	(15)	$x(yx) \approx xy$	(16)	$x1 \approx x$
(17)	$y+(y+x) \approx x$	(18)	$x(xy) \approx xy$	(19)	$0x \approx 0$

(where all $\mathcal{R}_1$-joinable critical pairs, like $x+(x+0) \approx 0$ or $x0 \approx y0$, are omitted). We obtain $\mathcal{E}_1 = \mathcal{E}_0 \cup \mathrm{Ext}_{\mathsf{o}}(\mathcal{E}_0)$. Now $\mathfrak{R}_{\mathsf{o}}(\mathcal{E}_1)$ may contain the TRS $\mathcal{R}_2$ consisting of the rules $(1),(3),(4),(5),(7),\ldots,(10),(13),\ldots,(19)$. This TRS is LPO-terminating, so there is a ground-total reduction order $>$ that contains $\rightarrow_{\mathcal{R}_2}$. We have $\mathcal{E}_1{\downarrow}_{\mathcal{R}_2} = \{(2),(6),(11),(12)\}$, and it can be shown that for $\mathcal{E} = \mathcal{E}_1{\downarrow}_{\mathcal{R}_2}$ the system $\mathcal{R}_2 \cup \mathcal{E}_>$ is ground complete. Despite its simplicity, neither WM [1] nor E [28] or Vampire [19] succeed on this example.

4.1 Theorem Proving

Next the approach is extended to purely equational theorem proving: Given a set of equations $\mathcal{E}_0$ and a goal equation $s \approx t$ as input, the aim is to decide whether $s \leftrightarrow^*_{\mathcal{E}_0} t$ holds. Let $\mathrm{Ext}_{\mathsf{g}}$ be a binary function on ESs such that $\mathrm{Ext}_{\mathsf{g}}(\mathcal{G},\mathcal{E}) \subseteq \leftrightarrow^*_{\mathcal{E}\cup\mathcal{G}} \setminus \leftrightarrow^*_{\mathcal{E}}$ for all ESs $\mathcal{E}$ and $\mathcal{G}$. In our implementation, $\mathrm{Ext}_{\mathsf{g}}(\mathcal{G},\mathcal{E})$ consists of extended critical pairs between an equation in $\mathcal{G}$ and an equation in $\mathcal{E}$. The following relation φ_{g} maps a pair of ESs $\mathcal{E}$ and $\mathcal{G}$ to YES or NO.

▶ **Definition 16.** *Given an ES $\mathcal{E}_0$, an initial ground goal $s_0 \approx t_0$ and ESs $\mathcal{E}$ and $\mathcal{G}$, let*

$$\varphi_{\mathsf{g}}(\mathcal{E},\mathcal{G}) = \begin{cases} \text{YES} & \text{if } s \downarrow_{\mathcal{R}\cup\mathcal{E}_>} t \text{ for some } s \approx t \in \mathcal{G} \text{ and } \mathcal{R} \in \mathfrak{R}_{\mathsf{o}}(\mathcal{E}), \\ \text{NO} & \text{if } \mathcal{R} \cup (\mathcal{E}{\downarrow}_{\mathcal{R}})_> \text{ is a ground complete presentation of } \mathcal{E}_0 \\ & \text{but } s_0 \not\downarrow_{\mathcal{R}\cup\mathcal{E}_>} t_0, \text{ for some } \mathcal{R} \in \mathfrak{R}_{\mathsf{o}}(\mathcal{E}), \text{ and} \\ \varphi_{\mathsf{g}}(\mathcal{E}\cup\mathcal{E}',\mathcal{G}\cup\mathcal{G}') & \text{for } \mathcal{G}' = \mathrm{Ext}_{\mathsf{g}}(\mathcal{G},\mathcal{R}\cup\mathcal{E}) \text{ and } \mathcal{E}' = \mathrm{Ext}_{\mathsf{o}}(\mathcal{E}). \end{cases}$$

For a set of input equations $\mathcal{E}_0$ and an initial goal $s_0 \approx t_0$, a maximal ordered completion procedure can then be run on the tuple $(\mathcal{E}_0, \{s_0 \approx t_0\})$. Note that the parameter $\mathcal{G}$ of φ_{g} denotes a disjunction of goals, not a conjunction. Due to the declarative nature of φ_{g} the following correctness result is straightforward.

▶ **Lemma 17.** *Let $\mathcal{E}_0$ be an ES and $s_0 \approx t_0$ be a ground goal. If $\varphi_{\mathsf{g}}(\mathcal{E}_0, \{s_0 \approx t_0\})$ is defined then $\varphi_{\mathsf{g}}(\mathcal{E}_0, \{s_0 \approx t_0\}) = \text{YES}$ if and only if $s_0 \leftrightarrow^*_{\mathcal{E}_0} t_0$.*

▶ **Example 18.** The conditional confluence tool ConCon [29] interfaces equational theorem provers like MædMax to show infeasibility of conditional critical pairs, which can be used to prove confluence of conditional TRSs. Here a conditional critical pair is called *infeasible*

if the involved conditions $s_1 \approx t_1, \ldots, s_n \approx t_n$ do not admit a substitution σ such that $s_i\sigma \leftrightarrow^*_{\mathcal{E}_0} t_i\sigma$ for all i. For example, ConCon encounters for Cops #340 the axioms $\mathcal{E}_0$:

$$\mathsf{f}(x_1, y_1) \approx \mathsf{g}(z_1) \qquad \mathsf{f}(x_1, \mathsf{h}(y_1)) \approx \mathsf{g}(z_1)$$

and the conditions $x_1 \approx y_1$, $\mathsf{h}(x_2) \approx y_1$, $x_1 \approx y_2$, $x_2 \approx y_2$. Let $s \approx t$ be the goal equation $\mathsf{conds}(x_1, \mathsf{h}(x_2), x_1, x_2) \approx \mathsf{conds}(y_1, y_1, y_2, y_2)$, using a fresh symbol conds. In order to decide whether there is a substitution σ such that $s\sigma \leftrightarrow^*_{\mathcal{E}_0} t\sigma$ holds, a common trick is used [6]: existence of such a σ can be refuted if the (ground) goal $\mathsf{true} \approx \mathsf{false}$ is not entailed by $\mathcal{E}_0$ extended with the following two equations:

$$\mathsf{eq}(x, x) \approx \mathsf{true} \quad (1) \qquad \mathsf{eq}(\mathsf{conds}(x_1, h(x_2), x_1, x_2), \mathsf{conds}(y_1, y_1, y_2, y_2)) \approx \mathsf{false} \quad (2)$$

For this extended ES $\mathcal{E}_0'$ a maximal ordered procedure call $\varphi_\mathsf{g}(\mathcal{E}_0', \{\mathsf{true} \approx \mathsf{false}\})$ can result in the answer NO immediately because a ground complete system exists, consisting of the two rewrite rules obtained when orienting (1) and (2) from left to right plus the following (unoriented) equations:

$$\mathsf{f}(x_1, y_1) \approx \mathsf{g}(z_1) \qquad \mathsf{f}(x_1, y_1) \approx \mathsf{f}(x_2, y_2) \qquad \mathsf{g}(x_1) \approx \mathsf{g}(y_1)$$

Both true and false are in normal form with respect to this system, so no suitable σ exists.

4.2 Completeness for Linear Systems

We conclude this section with a completeness result. A natural question in the context of completion is whether a complete system can be found by a completion procedure whenever it exists. For standard completion, it is well known that this is not the case: for example, the ES consisting of the equations $\mathsf{f}(x) \approx \mathsf{f}(\mathsf{a})$ and $\mathsf{f}(\mathsf{b}) \approx \mathsf{b}$ cannot be completed by Knuth-Bendix completion, or (standard) maximal completion if $\mathrm{Ext}(\mathcal{E}) \subseteq \bigcup_{\mathcal{R} \in \mathfrak{R}(\mathcal{E})} \mathrm{CP}(\mathcal{R})$. Nevertheless a complete presentation is given by the TRS $\{\mathsf{f}(x) \to \mathsf{b}\}$ [16].

For ordered completion, two sufficient conditions are known to answer this question in the positive: Bachmair, Dershowitz, and Plaisted showed that a complete system can always be found if the reduction order is ground total [6], and Devie proved that complete representations are invariably found for linear systems, irrespective of the order's totality [8].

Next, a completeness result for linear systems in the spirit of the result by Devie [8] is presented. To that end, the reduction order $>$ does not need to be ground total. In order to express that the reduction order leading to the presupposed completion system must be considered by the procedure, the function $\mathfrak{R}$ is said to *support* a reduction order $>$ if $\mathfrak{R}(\mathcal{E})$ contains a maximal TRS $\mathcal{R}$ such that $\mathcal{R} \subseteq >$, for all ESs $\mathcal{E}$.

Devie's notion of *linear overlaps* refers to extended overlaps which satisfy $\ell_1 > r_1$ and $r_2 \not> \ell_2$, or $\ell_2 > r_2$ and $r_1 \not> \ell_1$. Critical pairs originating from such overlaps are called *linear critical pairs*, and the set of all linear critical pairs formed using equations in $\mathcal{E}$ is denoted by $\mathsf{LCP}_>(\mathcal{E})$. A TRS $\mathcal{R}$ is called *reduced* if for all rules $\ell \to r$ in $\mathcal{R}$ both $r \in \mathsf{NF}(\mathcal{R})$ and $\ell \in \mathsf{NF}(\mathcal{R} \setminus \{\ell \to r\})$ hold.

▶ **Theorem 19.** *Let $\mathcal{E}_0$ be a linear ES which admits a complete and reduced presentation as the TRS $\mathcal{C}$ such that $\mathcal{C} \subseteq >$. Suppose moreover that $\mathfrak{R}$ supports $>$, $\mathrm{Ext}_\mathsf{o}(\mathcal{E})$ is linear whenever $\mathcal{E}$ is linear, and $\mathrm{Ext}_\mathsf{o}(\mathcal{E}) \supseteq \bigcup_{\mathcal{R} \in \mathfrak{R}(\mathcal{E})} \mathsf{LCP}_>(\mathcal{R} \cup \mathcal{E})$ for all ESs $\mathcal{E}$.*

Then $\varphi_\mathsf{o}(\mathcal{E}_0)$ is defined and constitutes a complete TRS.

Proof. Let $\mathcal{E}_0, \mathcal{E}_1, \mathcal{E}_2, \ldots$ be a maximal ordered completion sequence, and $\mathcal{S}_i$ denote the TRS $\mathcal{E}_{i>}$. It can be assumed that $\mathcal{E}_i$ is linear for all $i \geqslant 0$, because $\mathcal{E}_0$ is linear and Ext_o is supposed to preserve linearity.

Consider a cost function c defined on equation steps as follows: for $\ell \approx r \in \mathcal{E}_i$, let $c(s = C[\ell\sigma] \leftrightarrow_{\ell \approx r} C[r\sigma] = t)$ be $\{t\}$ if $\ell\sigma > r\sigma$, $\{s\}$ if $r\sigma > \ell\sigma$, and $\{s,t\}$ otherwise. This measure is extended to conversions $P\colon t_0 \leftrightarrow t_1 \leftrightarrow \ldots \leftrightarrow t_n$ by defining $c(P)$ as the multiset union $\bigcup_{0 \leqslant i < n} c(t_i \leftrightarrow t_{i+1})$. In the sequel $P \gg Q$ is written to abbreviate $c(P) >_{\mathsf{mul}} c(Q)$.

Consider a rule $\ell \to r$ in $\mathcal{C}$. As $\mathcal{C}$ is assumed to be a complete presentation of $\mathcal{E}_0$, there is a conversion $\ell \leftrightarrow^*_{\mathcal{E}_0} r$. According to Lemma 12, also $\ell \leftrightarrow^*_{\mathcal{E}_i} r$ holds for all $i \geqslant 0$. Let $P^i_{\ell\to r}$ be fixed conversions $\ell \leftrightarrow^*_{\mathcal{E}_i} r$ which are minimal with respect to $\gg$, for all $i \geqslant 0$.

We show that for every i, if $P^i_{\ell\to r}$ is not of the form $\ell \to^*_{\mathcal{S}_i} r$ then there is a conversion $P^{i+1}_{\ell\to r}$ which has fewer steps and satisfies $P^i_{\ell\to r} \gg P^{i+1}_{\ell\to r}$. Note that all conversions $\ell \leftrightarrow^*_{\mathcal{E}_i} r$ must have at least one step: otherwise, we would have $\ell = r$, which contradicts $\mathcal{C} \subseteq >$ because $>$ is well-founded.

Let $P^i_{\ell\to r}$ be a minimal conversion $\ell \leftrightarrow^*_{\mathcal{E}_i} r$. Since it has at least one step, we can assume some term r' such that $P^i_{\ell\to r}$ has the form $\ell \leftrightarrow^*_{\mathcal{E}_i} r' \leftrightarrow_{\mathcal{E}_i} r$, and $r' \neq r$. Note that the last step $r' \leftrightarrow_{\mathcal{E}_i} r$ must satisfy $r' > r$: By conversion equivalence, we must have $r' \leftrightarrow^*_{\mathcal{C}} r$. Since $\mathcal{C}$ is complete, $r' \downarrow_{\mathcal{C}} r$ holds. Because $\mathcal{C}$ is also reduced, the term r is irreducible, so we have $r' \to^*_{\mathcal{C}} r$. By the above assumption that $r' \neq r$ this means that $r' \to^+_{\mathcal{C}} r$, which implies $r' > r$ because $\mathcal{C} \subseteq >$. So the equation step $r' \leftrightarrow_{\mathcal{E}_i} r$ is an ordered rewrite step $r' \to_{\mathcal{S}_i} r$.

If $\ell \leftrightarrow^*_{\mathcal{E}_i} r$ is not of the form $\ell \to^*_{\mathcal{S}_i} r$ then it must therefore contain a peak involving non-$\mathcal{S}_i$ step followed by an $\mathcal{S}_i$ step, that is, a peak of the form

$$Q\colon s \xleftrightarrow[\ell_1 \approx r_1, \sigma]{} u \xrightarrow[\ell_2 \approx r_2, \sigma]{} t$$

for some terms s, t, and u, equations $\ell_1 \approx r_1$, $\ell_2 \approx r_2 \in \mathcal{E}_i$, and a substitution σ such that $\ell_1\sigma \not> r_1\sigma$ but $\ell_2\sigma > r_2\sigma$, so $\ell_2\sigma \to r_2\sigma \in \mathcal{S}_i$. Note that $c(Q) = \{s,u,t\}$.

(a) If $\ell_1 \approx r_1$ and $\ell_2 \approx r_2$ form a proper overlap then $s \leftrightarrow_{\mathsf{LCP}_>(\mathcal{E}_i)} t$ because $\ell_1\sigma \not> r_1\sigma$ and $\ell_2\sigma > r_2\sigma$. By assumption $\mathsf{LCP}_>(\mathcal{E}_i) \subseteq \mathcal{E}_{i+1}$. Hence there is a conversion $P^{i+1}_{\ell\to r}\colon \ell \leftrightarrow^*_{\mathcal{E}_{i+1}} r$ where Q is replaced by $Q'\colon s \leftrightarrow_{\mathcal{E}_{i+1}} t$ and $c(Q) >_{\mathsf{mul}} \{s,t\} \geqslant_{\mathsf{mul}} c(Q')$. Moreover, $P^{i+1}_{\ell\to r}$ has fewer steps than $P^i_{\ell\to r}$.

(b) Suppose $\ell_1 \approx r_1$ and $\ell_2 \approx r_2$ occur in parallel. Then the two steps can be swapped, so there is a term v which allows for the conversion $Q'\colon s \to_{\ell_2\sigma \to r_2\sigma} v \leftrightarrow_{\ell_1\sigma \approx r_1\sigma} t$. This contradicts the assumption that $P^i_{\ell\to r}$ was minimal: we have $c(Q) >_{\mathsf{mul}} \{v,v,t\} = c(Q')$ because $s > v$.

(c) Similarly, if $\ell_1 \approx r_1$ and $\ell_2 \approx r_2$ form a variable overlap then because $\mathcal{E}_i$ is linear there is a term v such that there is a conversion $Q\colon s \to^=_{\ell_2\sigma \to r_2\sigma} v \leftrightarrow^=_{\ell_1\sigma \approx r_1\sigma} t$. But this again contradicts minimality of $P^i_{\ell\to r}$ because $C(Q) >_{\mathsf{mul}} \{v,v,t\} \geqslant_{\mathsf{mul}} c(Q')$ due to $s > v$.

Let k be the maximal number of steps of $P^0_{\ell\to r}$, for $\ell \to r \in \mathcal{C}$. The above argument shows that $\ell \to^*_{\mathcal{S}_k} r$ holds for all $\ell \to r \in \mathcal{C}$. Hence we have $\mathsf{NF}(\mathcal{S}_k) \subseteq \mathsf{NF}(\mathcal{C})$.

Let $\mathcal{S}$ be the TRS $\mathcal{R} \cup (\mathcal{E}_k{\downarrow_{\mathcal{R}}})_>$. Any term reducible by $\mathcal{S}_k$ must also be reducible in $\mathcal{S}$, which implies $\mathsf{NF}(\mathcal{S}) \subseteq \mathsf{NF}(\mathcal{S}_k)$ and hence $\mathsf{NF}(\mathcal{S}) \subseteq \mathsf{NF}(\mathcal{C})$. Since moreover $\mathcal{R} \cup \mathcal{E}_k \subseteq \leftrightarrow^*_{\mathcal{C}}$ implies $\to_{\mathcal{S}_k} \subseteq \leftrightarrow^*_{\mathcal{C}}$ and $\mathcal{S}$ is terminating because $\mathcal{S} \subseteq >$, the TRS $\mathcal{S}$ is complete according to [11, Lemma 31]. ◀

Note that the above proof implies a bound on the number of iterations needed to derive a complete system, namely the number of $\mathcal{E}_0$-steps required for conversions of the rules in the complete system $\mathcal{C}$. Naturally, due to incompleteness of implementations, this bound cannot be kept up in practice.

▶ **Example 20.** Consider the linear ES $\mathcal{E}$ consisting of the following three equations:

$$\mathsf{f}(\mathsf{a},\mathsf{i}(x)) \approx \mathsf{f}(\mathsf{b},\mathsf{b}) \qquad \mathsf{g}(\mathsf{b},x) \approx \mathsf{g}(\mathsf{a},\mathsf{a}) \qquad \mathsf{f}(\mathsf{a},x) \approx \mathsf{f}(\mathsf{a},y)$$

The TRS $\mathcal{R} = \{\mathsf{f}(\mathsf{a},x) \to \mathsf{f}(\mathsf{b},\mathsf{b}),\ \mathsf{g}(\mathsf{b},x) \to \mathsf{g}(\mathsf{a},\mathsf{a})\}$ is terminating and confluent, as is easily checked by state-of the art tools. We also have $\mathcal{E}_0 \subseteq \downarrow_{\mathcal{R}}$, and from the conversion

$$\mathsf{f}(\mathsf{a},x) \leftrightarrow \mathsf{f}(\mathsf{a},\mathsf{i}(x)) \leftrightarrow \mathsf{f}(\mathsf{b},\mathsf{b}) \tag{2}$$

we can conclude $\leftrightarrow^*_{\mathcal{E}_0} = \leftrightarrow^*_{\mathcal{R}}$, so $\mathcal{R}$ is a complete presentation of $\mathcal{E}_0$. By Theorem 19, maximal ordered completion supporting $> = \to^+_{\mathcal{R}}$ will succeed with a complete system, and according to the bound derived in the proof, this takes at most two iterations since (2) has two steps.

5 Normalized Completion

Many algebraic theories like groups and rings feature associative and commutative operators. However, since the commutativity equation cannot be oriented into a terminating rewrite rule, such theories cannot be handled by standard Knuth-Bendix completion. This triggered the development of dedicated completion calculi that can deal with such cases [23, 12].

Various generalizations have been proposed to extend completion to different algebraic theories, apart from plain AC. A version for general theories $\mathcal{T}$ has been proposed in [12, 4], provided that $\mathcal{T}$ admits finitary unification and the subterm ordering modulo $\mathcal{T}$ is well-founded. Constrained completion [13] constitutes an attempt to overcome these restrictions on the theory, it admits for instance completion modulo AC with a unit element (ACU). However, it excludes other theories such as Abelian groups.

Normalized completion [21, 22, 34] can be seen as the last result in this line of research. It has three advantages over earlier methods. (1) It allows completion modulo any theory $\mathcal{T}$ that can be represented as an AC-complete rewrite system $\mathcal{S}$. (2) Critical pairs need not be computed for the theory $\mathcal{T}$, which may not be finitary or even have a decidable unification problem. Instead, any theory between AC and $\mathcal{T}$ can be used. (3) The AC-compatible reduction order used to establish termination need not be compatible with $\mathcal{T}$. This is beneficial for theories like ACU where no $\mathcal{T}$-compatible simplification order exists.

▶ **Example 21.** Consider an Abelian group with AC operator $\cdot$ and an endomorphism f as described by the following three equations:

$$\mathsf{e} \cdot x \approx x \qquad \mathsf{i}(x) \cdot x \approx \mathsf{e} \qquad \mathsf{f}(x \cdot y) \approx \mathsf{f}(x) \cdot \mathsf{f}(y)$$

together with ACRPO [24] with precedence $\mathsf{f} > \mathsf{i} > \cdot > \mathsf{e}$. Using AC completion, or equivalently normalized completion with respect to $\mathcal{S} = \varnothing$, one obtains the following AC complete TRS $\mathcal{R}_{\mathrm{AC}}$:

$$\begin{array}{ccc} \mathsf{e} \cdot x \to x & \mathsf{i}(x) \cdot x \to \mathsf{e} & \mathsf{i}(\mathsf{e}) \to \mathsf{e} \\ \mathsf{i}(\mathsf{i}(x)) \to x & \mathsf{i}(x \cdot y) \to \mathsf{i}(x) \cdot \mathsf{i}(y) & \mathsf{f}(x \cdot y) \to \mathsf{f}(x) \cdot \mathsf{f}(y) \\ \mathsf{f}(\mathsf{e}) \to \mathsf{e} & \mathsf{f}(\mathsf{i}(x)) \to \mathsf{i}(\mathsf{f}(x)) & \end{array}$$

Alternatively, one can perform normalized completion with respect to an AC complete representation of Abelian groups, like for example the following TRS $\mathcal{S}_{\mathsf{G}}$ [3]:

$$\mathsf{e} \cdot x \to x \qquad \mathsf{i}(x) \cdot x \to \mathsf{e} \qquad \mathsf{i}(\mathsf{e}) \to \mathsf{e} \qquad \mathsf{i}(\mathsf{i}(x)) \to x \qquad \mathsf{i}(x \cdot y) \to \mathsf{i}(x) \cdot \mathsf{i}(y)$$

Note that $\mathcal{S}_{\mathsf{G}} \subseteq >$. Normalized completion with respect to $\mathcal{S}_{\mathsf{G}}$ results in the TRS $\mathcal{R}_{\mathsf{G}}$:

$$\mathsf{f}(x \cdot y) \to \mathsf{f}(x) \cdot \mathsf{f}(y) \qquad \mathsf{f}(\mathsf{e}) \to \mathsf{e} \qquad \mathsf{f}(\mathsf{i}(x)) \to \mathsf{i}(\mathsf{f}(x))$$

Before proposing a maximal normalized completion procedure, we recall some concepts and notations related to AC rewriting and normalized rewriting.

AC Rewriting and Unification. A TRS $\mathcal{R}$ *terminates modulo* AC whenever the relation $\to_{\mathcal{R}/\mathrm{AC}}$ is well-founded. To establish AC termination we will consider AC-compatible reduction orders $>$, i.e., reduction orders that satisfy $\leftrightarrow^*_{\mathrm{AC}} \cdot > \cdot \leftrightarrow^*_{\mathrm{AC}} \subseteq >$. The TRS $\mathcal{R}$ is *complete modulo* AC if it terminates modulo AC and the relation $\leftrightarrow^*_{\mathrm{AC}\,\cup\,\mathcal{R}}$ coincides with $\to^*_{\mathcal{R}/\mathrm{AC}} \cdot \leftrightarrow^*_{\mathrm{AC}} \cdot \leftarrow^*_{\mathcal{R}/\mathrm{AC}}$. It is an AC-*complete presentation* of an ES $\mathcal{E}$ if $\mathcal{R}$ is AC complete and $\leftrightarrow^*_{\mathcal{E}\cup\mathrm{AC}} = \leftrightarrow^*_{\mathcal{R}\cup\mathrm{AC}}$.

Let $\mathcal{L}$ be a theory with finitary and decidable unification problem. A substitution σ constitutes an $\mathcal{L}$-*unifier* of two terms s and t if $s\sigma \leftrightarrow^*_{\mathcal{L}} t\sigma$ holds. An $\mathcal{L}$-*overlap* is a quadruple $\langle \ell_1 \to r_1, p, \ell_2 \to r_2 \rangle_\Sigma$ consisting of rewrite rules $\ell_1 \to r_1$, $\ell_2 \to r_2$, a position $p \in \mathcal{P}\mathrm{os}_{\mathcal{F}}(\ell_2)$, and a complete set Σ of $\mathcal{L}$-unifiers of $\ell_2|_p$ and ℓ_1. Then $\ell_2[r_1]_p\sigma \approx r_2\sigma$ constitutes an $\mathcal{L}$-*critical pair* for every $\sigma \in \Sigma$. For two sets of rewrite rules $\mathcal{R}_1$ and $\mathcal{R}_2$, we also write $\mathrm{CP}_{\mathcal{L}}(\mathcal{R}_1, \mathcal{R}_2)$ for the set of all $\mathcal{L}$-critical pairs emerging from an overlap where $\ell_1 \to r_1 \in \mathcal{R}_1$ and $\ell_2 \to r_2 \in \mathcal{R}_2$, and $\mathrm{CP}_{\mathcal{L}}(\mathcal{R}_1)$ for the set of all $\mathcal{L}$-critical pairs such that $\ell_1 \to r_1$, $\ell_2 \to r_2 \in \mathcal{R}_1$.

We assume there is a fixed set of AC symbols $\mathcal{F}_{\mathrm{AC}} \subseteq \mathcal{F}$. For a rewrite rule $\ell \to r$ with $+ \in \mathcal{F}_{\mathrm{AC}}$ the notation $(\ell \to r)^e$ refers to the *extended rule* $\ell + x \to r + x$, where $x \in \mathcal{V}$ is fresh. The TRS $\mathcal{R}^e$ contains all rules in $\mathcal{R}$ plus all extended rules $\ell + x \to r + x$ such that $\ell \to r \in \mathcal{R}$ [3].

Normalized Rewriting. We define normalized rewriting as in [22] but use a different notation to distinguish it from the common notation for rewriting modulo. Let $\mathcal{T}$ be a theory which has an AC-complete presentation as a TRS $\mathcal{S}$.

Two terms s and t admit an $\mathcal{S}$-*normalized* $\mathcal{R}$-*rewrite step* if

$$s \xrightarrow[\mathcal{S}/\mathrm{AC}]{!} \cdot \xleftrightarrow[\mathrm{AC}]{*} \cdot \xrightarrow[\ell\to r]{p} \cdot \xleftrightarrow[\mathrm{AC}]{*} t \tag{3}$$

for some rule $\ell \to r$ in $\mathcal{R}$ and position p. We abbreviate (3) by $s \to^p_{\ell\to r\backslash\mathcal{S}} t$ and write $s \to_{\mathcal{R}\backslash\mathcal{S}} t$ if $s \to^p_{\ell\to r\backslash\mathcal{S}} t$ for a rule $\ell \to r$ in $\mathcal{R}$ and position p. Let $>$ be an AC-compatible reduction order such that $\mathcal{S} \subseteq >$. For any set of rewrite rules $\mathcal{R}$ satisfying $\mathcal{R} \subseteq >$ the normalized rewrite relation $\to_{\mathcal{R}\backslash\mathcal{S}}$ is well-founded [21, 22], so we can consider equational proofs of the form $s \to^!_{\mathcal{R}\backslash\mathcal{S}} \cdot \leftrightarrow^*_{\mathcal{T}} \cdot \leftarrow^!_{\mathcal{R}\backslash\mathcal{S}} t$. These normal form proofs play a special role and are called *normalized rewrite proofs*. Because $\mathcal{S}$ is AC-complete for $\mathcal{T}$, any such proof can be transformed into a proof $s \Downarrow_{\mathcal{R}\backslash\mathcal{S}} t$, where $\Downarrow_{\mathcal{R}\backslash\mathcal{S}}$ abbreviates the relation

$$\xrightarrow[\mathcal{R}\backslash\mathcal{S}]{!} \cdot \xrightarrow[\mathcal{S}/\mathrm{AC}]{!} \cdot \xleftrightarrow[\mathrm{AC}]{*} \cdot \xleftarrow[\mathcal{S}/\mathrm{AC}]{!} \cdot \xleftarrow[\mathcal{R}\backslash\mathcal{S}]{!}$$

A TRS $\mathcal{R}$ is an $\mathcal{S}$-*complete presentation* of a set of equations $\mathcal{E}$ if $\to_{\mathcal{R}\backslash\mathcal{S}}$ is terminating and the relations $\leftrightarrow^*_{\mathcal{E}\,\cup\,\mathcal{T}}$ and $\to^!_{\mathcal{R}\backslash\mathcal{S}} \cdot \leftrightarrow^*_{\mathcal{T}} \cdot \leftarrow^!_{\mathcal{R}\backslash\mathcal{S}}$, hence $\Downarrow_{\mathcal{R}\backslash\mathcal{S}}$, coincide.

In the remainder of this section we assume that $\mathfrak{R}_{\mathcal{S}}(\mathcal{E})$ is a finite set of rewrite systems $\mathcal{R}$ such that $\mathcal{R} \cup \mathcal{S}$ is AC terminating, for all ESs $\mathcal{E}$. Moreover, let the function $\mathrm{Ext}_{\mathcal{S}}$ satisfy $\mathrm{Ext}_{\mathcal{S}}(\mathcal{E}) \subseteq \leftrightarrow^*_{\mathrm{AC}\,\cup\,\mathcal{S}\,\cup\,\mathcal{E}}$ for all ESs $\mathcal{E}$. We write $\mathrm{CP}_{\mathcal{S}}(\mathcal{R})$ for the set of critical pairs

$$\mathrm{CP}_{\mathcal{L}}(\mathcal{R}^e) \cup \mathrm{CP}_{\mathrm{AC}}(\mathcal{S}^e, \mathcal{R}^e) \cup \mathrm{CP}_{\mathrm{AC}}(\mathcal{R}^e, \mathcal{S}^e)$$

▶ **Definition 22.** *Given a set of input equalities $\mathcal{E}_0$ and an ES $\mathcal{E}$, let*

$$\varphi_{\mathcal{S}}(\mathcal{E}) = \begin{cases} \mathcal{R} & \text{if } \mathcal{R} \in \mathfrak{R}_{\mathcal{S}}(\mathcal{E}) \text{ such that } \mathrm{CP}_{\mathcal{S}}(\mathcal{R}) \cup \mathcal{E}_0 \subseteq \Downarrow_{\mathcal{R}\backslash\mathcal{S}} \\ \varphi_{\mathcal{S}}(\mathcal{E} \cup \mathrm{Ext}_{\mathcal{S}}(\mathcal{E})) & \text{otherwise.} \end{cases}$$

The proof of the following correctness statement is a straightforward adaptation of the respective result for standard completion (Lemma 4).

▶ **Lemma 23.** *If $\varphi_{\mathcal{S}}(\mathcal{E})$ is defined then it is an $\mathcal{S}$-complete presentation of $\mathcal{E}_0$.*

Proof. Suppose $\varphi_{\mathcal{S}}(\mathcal{E}_0) = \mathcal{R}$, so $\mathcal{R} \cup \mathcal{S}$ is AC terminating since it was returned by $\mathfrak{R}_{\mathcal{S}}$. Because of $\mathrm{CP}_{\mathcal{S}}(\mathcal{R}) \subseteq \Downarrow_{\mathcal{R}\backslash\mathcal{S}}$ the TRS $\mathcal{R}$ is $\mathcal{S}$-complete according to the results by Marché [22].

Let $\mathcal{E}_1, \dots, \mathcal{E}_k$ be a sequence of normalized maximal completion, that is $\mathcal{E}_{i+1} = \mathcal{E}_i \cup \mathrm{Ext}_{\mathcal{S}}(\mathcal{E}_i)$ for all $1 \leqslant i < k$ and there is some $\mathcal{R} \in \mathfrak{R}_{\mathcal{S}}(\mathcal{E}_k)$ such that $\mathrm{CP}_{\mathcal{S}}(\mathcal{R}) \cup \mathcal{E}_0 \subseteq \Downarrow_{\mathcal{R}\backslash\mathcal{S}}$. A simple induction argument using the global assumption that $\mathrm{Ext}_{\mathcal{S}}(\mathcal{E}) \subseteq \leftrightarrow^*_{\mathrm{AC}\,\cup\,\mathcal{S}\,\cup\,\mathcal{E}}$ for all ESs $\mathcal{E}$ shows that $\mathcal{E}_k \subseteq \leftrightarrow^*_{\mathrm{AC}\,\cup\,\mathcal{S}\,\cup\,\mathcal{E}_0}$. Since $\mathcal{R}$ is over $\mathcal{E}_k$, also $\leftrightarrow^*_{\mathcal{R}} \subseteq \leftrightarrow^*_{\mathrm{AC}\,\cup\,\mathcal{S}\,\cup\,\mathcal{E}_0}$ holds. Conversely, $\mathcal{E}_0 \subseteq \Downarrow_{\mathcal{R}\backslash\mathcal{S}}$ is assumed. So $\mathcal{R}$ is an $\mathcal{S}$-complete presentation of $\mathcal{E}_0$. ◀

The maximal normalized completion implementation in MædMax can for instance complete the ES in Example 21 with respect to both AC (so $\mathcal{S} = \varnothing$) or group theory (using $\mathcal{S}_{\mathsf{G}}$).

6 Implementation

In this section we briefly summarize an implementation of the discussed variants of maximal completion in the tool MædMax [36]. MædMax is implemented in OCaml and available as a command-line tool as well as via a web interface, on the accompanying website also example input can be found.[1] Input problems can be submitted in the TPTP [31] as well as the trs format.[2] The tool supports standard maximal completion, maximal ordered completion and theorem proving, as well as normalized completion. However, many modules are used for all of these modes. For the former, MædMax incorporates the extended Maxcomp version [25] which supports advanced termination techniques like dependency pairs.

In the following paragraphs we comment on the implementation of the three components corresponding to the main steps in maximal completion: (1) finding (AC) terminating TRSs, (2) success checks, and (3) selection of new equations and goals.

Finding rewrite systems. In order to find (AC) terminating rewrite systems that play the role of $\mathfrak{R}(\mathcal{E})$ and $\mathfrak{R}_{\mathcal{S}}(\mathcal{E})$, respectively, MædMax adheres to the basic approach of Maxcomp [17] in that it solves optimization problems by means of a maxSAT/maxSMT solver. The objective of this optimization can be to (a) maximize the number of oriented equations as done in Maxcomp, or (b) the equations in $\mathcal{E}$ that are reducible, or to (c) minimize the number of rules or (d) the number of critical pairs. These optimization targets can also be combined, and completeness requirements as described in [25] can be added. Strategy (b) in combination with (c) has proved to be particularly useful, because it prefers small TRSs which can simplify many equations. This is especially beneficial in presence of AC symbols, where many rewrite rules and hence many critical pairs can drastically impact performance.

In order to guarantee termination of the resulting system, SMT encodings of termination techniques are used. These are LPO, KBO, and linear polynomials for ordered completion, where a ground-total reduction order is desired. For standard completion, MædMax additionally supports dependency pairs, a dependency graph approximation, and argument filterings for LPO and KBO, as described earlier [25]. These techniques can also be combined in a strategy involving sequential composition and choice. As a means to ensure AC termination, ACRPO is encoded [24]. The supported SMT solvers are Yices 1.0 [10] and Z3 [7].

[1] http://cl-informatik.uibk.ac.at/software/maedmax/
[2] https://www.lri.fr/~marche/tpdb/format.html

Success checks. For standard and normalized completion, it is straightforward to check whether all critical pairs are joinable. In the latter case, MædMax only supports AC critical pairs. To conclude ground confluence, our tool supports the criterion of [33].

Selection. The extension functions Ext, $\mathrm{Ext_g}$, and $\mathrm{Ext}_{\mathcal{S}}$ are implemented to add a subset of (extended, AC) critical pairs among rules in $\mathcal{R}$, and equations/goal for the case of ordered completion. In any case the selected equations get reduced to $\mathcal{R}$-normal form before they are added. MædMax severely limits the number of critical pairs that are added in every iteration to confine the exponential blowup. The selection heuristic prefers small equations and old, but not yet reducible equations.

Furthermore, MædMax can output equational (dis)proofs and ground completion proofs in a format that can be validated by the proof checker CeTA [30]. Further implementation details and evaluations on standard benchmark sets can be found in [36, 25].

We conclude with a final example illustrating a practical application. The tool AQL[3] performs functorial data integration by means of a category-theoretic approach [27], taking advantage of (ground) completion. The following problem was communicated by the authors.

▶ **Example 24.** Consider two database tables yIsAL and yIsAW relating amphibians to land and water animals, respectively. The relationship between their entries are described by 400 ground equations over symbols $\mathsf{yIsAL}, \mathsf{yIsALL}, \mathsf{yIsAW}, \mathsf{yIsAWW}$ (which correspond to fields in the schemas) and 449 constants of the form $\mathsf{a}_i, \mathsf{w}_i, \mathsf{l}_i$ representing data items. The following six example equations may convey an impression:

$$\mathsf{yIsAW}(\mathsf{a}_1) \approx \mathsf{w}_{29} \qquad \mathsf{yIsAW}(\mathsf{a}_{78}) \approx \mathsf{w}_{16} \qquad \mathsf{yIsAW}(\mathsf{a}_{61}) \approx \mathsf{w}_{30}$$
$$\mathsf{yIsAL}(\mathsf{a}_{37}) \approx \mathsf{l}_{80} \qquad \mathsf{yIsAL}(\mathsf{a}_{84}) \approx \mathsf{l}_{6} \qquad \mathsf{yIsAL}(\mathsf{a}_{29}) \approx \mathsf{l}_{47}$$

In addition, the equation $\mathsf{yIsALL}(\mathsf{yIsAL}(x)) \approx \mathsf{yIsAWW}(\mathsf{yIsAW}(x))$ describes a mapping to a second database schema. A ground complete presentation of the entire system thus constitutes a representation of the data, translated to the second schema. MædMax discovers a complete presentation of 889 rules in less then 20 seconds, while AQL's internal completion prover fails. MædMax' automatic mode switches to linear polynomials for such systems with many symbols, which turned out to be faster than LPO or KBO in this situation.

7 Conclusion

This paper explored variants of maximal completion, corresponding to ordered and normalized completion. These methods have multiple advantages over earlier approaches:

- The reduction order, a notoriously critical parameter, need not be fixed in advance. This also holds for tools with an automatic mode such as RRL [15], but there it is unsound to change the order once it was fixed [26]. In contrast, no such problem occurs in maximal completion.
- Using maxSMT encodings, the choice of an ordering can be "steered" towards beneficial properties of the resulting system (e.g. to orient a maximal number of equations, to reduce a maximal number of equations, or to stimulate complete systems [25]).
- Maximal completion exploits the advantage of parallelization in that multiple reduction orders can be considered (by choosing multiple rewrite systems in every iteration). Theorem 19 shows that in the linear case any complete system for a supported ordering will be found. But at the same time rewriting and critical pair computation are shared among the processes corresponding to the different choices of an ordering.

[3] `http://categoricaldata.net/aql.html`

- Efficiency is gained by orienting multiple equations at the same time. Theorem 19 shows that this also admits a (theoretical) bound on the number of required iterations.
- Finally, the definitions and the corresponding proofs are concise and simple: neither proof orders [5] nor notions like peak or source decreasingness [11] are required.

Several directions for future work arise. First, we believe that also the completeness result for ground-total reduction orders carries over to maximal ordered completion [6]. The general case of completeness is still an open problem. Another interesting because practically relevant variant of completion operates on logically constrained rewrite systems (LCTRSs) [35]. Supporting maximal completion procedure for this setting might thus be a useful addition to MædMax. Maximal completion can be considered an approximation- and conflict-based approach: complete TRSs are overapproximated by terminating TRSs, and if a conflict (that is a non-joinable critical pair) is encountered, the approximation is refined. It would be interesting to investigate connections to other conflict-driven learning approaches such as lazy SMT solving or DPLL [9].

References

1 J. Avenhaus, T. Hillenbrand, and B. Löchner. On using ground joinable equations in equational theorem proving. *J. Symb. Comput.*, 36(1–2):217–233, 2003. doi:10.1016/S0747-7171(03)00024-5.

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

3 L. Bachmair. *Canonical Equational Proofs.* Progress in Theoretical Computer Science. Birkhäuser, 1991.

4 L. Bachmair and N. Dershowitz. Completion for Rewriting Modulo a Congruence. *Theor. Comput. Sci.*, 67(2,3):173–201, 1989. doi:10.1016/0304-3975(89)90003-0.

5 L. Bachmair, N. Dershowitz, and Jieh Hsiang. Orderings for Equational Proofs. In *Proc. 1st LICS*, pages 346–357, 1986.

6 L. Bachmair, N. Dershowitz, and D. A. Plaisted. Completion Without Failure. In H. Aït Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures*, volume 2 of *Rewriting Techniques*, pages 1–30. Academic Press, 1989. doi:10.1016/B978-0-12-046371-8.50007-9.

7 N. Bjørner. Taking Satisfiability to the Next Level with Z3. In *Proc. 6th IJCAR*, volume 7364 of *LNCS*, pages 1–8, 2012. doi:10.1007/978-3-642-31365-3_1.

8 H. Devie. Linear Completion. In *Proc. 2nd CTRS*, pages 233–245. Springer, 1991. doi:10.1007/3-540-54317-1_94.

9 V. D'Silva, L. Haller, and D. Kroening. Abstract Conflict Driven Learning. *ACM SIGPLAN Notices*, 48(1):143–154, 2013. doi:10.1145/2480359.2429087.

10 B. Dutertre and L. de Moura. A Fast Linear-Arithmetic Solver for DPLL(T). In *Proc. 18th CAV*, volume 4144 of *LNCS*, pages 81–94, 2006. doi:10.1007/11817963_11.

11 N. Hirokawa, A. Middeldorp, C. Sternagel, and S. Winkler. Infinite Runs in Abstract Completion. In *Proc. 2nd FSCD*, volume 84 of *LIPIcs*, pages 19:1–19:16, 2017. doi:10.4230/LIPIcs.FSCD.2017.19.

12 J.-P. Jouannaud and H. Kirchner. Completion of a Set of Rules Modulo a Set of Equations. *SIAM Journal of Computation*, 15(4):1155–1194, 1986. doi:10.1137/0215084.

13 J.-P. Jouannaud and C. Marché. Termination and Completion Modulo Associativity, Commutativity and Identity. *Theor. Comput. Sci.*, 104(1):29–51, 1992. doi:10.1016/0304-3975(92)90165-C.

14 D. Kapur, D. R. Musser, and P. Narendran. Only Prime Superpositions Need be Considered in the Knuth-Bendix Completion Procedure. *J. Symb. Comput.*, 6(1):19–36, 1988. doi:10.1016/S0747-7171(88)80019-1.

15 D. Kapur and H. Zhang. An overview of Rewrite Rule Laboratory (RRL). *Comput. Math. Appl.*, 29(2):91–114, 1995. doi:10.1016/0898-1221(94)00218-A.
16 D. Klein. *Equational Reasoning and Completion*. PhD thesis, Japan Advanced Institute of Science and Technology, 2012.
17 D. Klein and N. Hirokawa. Maximal Completion. In *Proc. 22nd RTA*, volume 10 of *LIPIcs*, pages 71–80, 2011. doi:10.4230/LIPIcs.RTA.2011.71.
18 D. E. Knuth and P. Bendix. Simple Word Problems in Universal Algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970. doi:10.1016/B978-0-08-012975-4.
19 L. Kovács and A. Voronkov. First-Order Theorem Proving and Vampire. In *Proc. 25th CAV*, volume 8044 of *LNCS*, pages 1–35, 2013. doi:10.1007/978-3-642-39799-8_1.
20 B. Löchner. A Redundancy Criterion Based on Ground Reducibility by Ordered Rewriting. In *Proc. 2nd IJCAR*, volume 3097 of *LNCS*, pages 45–59, 2004. doi:10.1007/978-3-540-25984-8_2.
21 C. Marché. Normalised Rewriting and Normalised Completion. In *LICS 1994*, pages 394–403. IEEE Computer Society, 1994. doi:10.1109/LICS.1994.316050.
22 C. Marché. Normalized Rewriting: An Alternative to Rewriting Modulo a Set of Equations. *J. Symb. Comput.*, 21(3):253–288, 1996. doi:10.1006/jsco.1996.0011.
23 G. E. Peterson and M. E. Stickel. Complete Sets of Reductions for Some Equational Theories. *J. ACM*, 28(2):233–264, 1981. doi:10.1145/322248.322251.
24 A. Rubio. A fully syntactic AC-RPO. *Inform. Comput.*, 178(2):515–533, 2002. doi:10.1006/inco.2002.3158.
25 H. Sato and S. Winkler. Encoding Dependency Pair Techniques and Control Strategies for Maximal Completion. In *Proc. 25th CADE*, volume 9195 of *LNCS*, pages 152–162, 2015. doi:10.1007/978-3-319-21401-6_10.
26 A. Sattler-Klein. About changing the ordering during Knuth-Bendix completion. In *Proc. 11th STACS*, volume 775 of *LNCS*, pages 175–186, 1994. doi:10.1007/3-540-57785-8_140.
27 P. Schultz and R. Wisnesky. Algebraic data integration. *J. Funct. Program.*, 27(e24):51 pages, 2017. doi:10.1017/S0956796817000168.
28 S. Schulz. System Description: E 1.8. In *Proc. 19th LPAR*, volume 8312 of *LNCS*, pages 735–743, 2013. doi:10.1007/978-3-642-45221-5_49.
29 T. Sternagel and A. Middeldorp. Conditional Confluence (System Description). In *Proc. RTA/TLCA 2014*, volume 8560 of *LNCS*, pages 456–465, 2014. doi:10.1007/978-3-319-08918-8_31.
30 T. Sternagel, S. Winkler, and H. Zankl. Recording Completion for Certificates in Equational Reasoning. In *Proc. 4th CPP*, pages 41–47, 2015. doi:10.1145/2676724.2693171.
31 G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts. *J. Autom. Reasoning*, 43(4):337–362, 2009. doi:10.1007/s10817-009-9143-8.
32 I. Wehrman and A. Stump. Mining Propositional Simplification Proofs for Small Validating Clauses. *Theor. Comput. Sci.*, 144(2):79–91, 2006. doi:10.1016/j.entcs.2005.12.008.
33 S. Winkler. A Ground Joinability Criterion for Ordered Completion. In *Proc. 6th IWC*, pages 45–49, 2017.
34 S. Winkler and A. Middeldorp. Normalized Completion Revisited. In *Proc. 24th RTA*, volume 21 of *LIPIcs*, pages 319–334, 2013. doi:10.4230/LIPIcs.RTA.2013319.
35 S. Winkler and A. Middeldorp. Completion for Logically Constrained Rewriting. In *Proc. 3rd FSCD*, volume 108 of *LIPIcs*, pages 30:1–30:18, 2018. doi:10.4230/LIPIcs.FSCD.2018.30.
36 S. Winkler and G. Moser. MaedMax: A Maximal Ordered Completion Tool. In *Proc. 9th IJCAR*, volume 10900 of *LNCS*, pages 472–480, 2018. doi:10.1007/978-3-319-94205-6_31.

Some Semantic Issues in Probabilistic Programming Languages

Hongseok Yang
School of Computing, KAIST, South Korea
hongseok.yang@kaist.ac.kr

Abstract

This is a slightly extended abstract of my talk at FSCD'19 about probabilistic programming and a few semantic issues on it. The main purpose of this abstract is to provide keywords and references on the work mentioned in my talk, and help interested audience to do follow-up study.

2012 ACM Subject Classification Theory of computation → Probabilistic computation; Theory of computation → Program semantics; Theory of computation → Denotational semantics; Mathematics of computing → Bayesian nonparametric models; Mathematics of computing → Bayesian computation

Keywords and phrases Probabilistic Programming, Denotational Semantics, Non-differentiable Models, Bayesian Nonparametrics, Exchangeability

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.4

Category Invited Talk

1 Introduction

Probabilistic programming [11, 9, 35, 8] refers to the idea of developing a programming language for writing and reasoning about probabilistic models from machine learning and statistics. Such a language comes with the implementation of several generic inference algorithms that answer various queries about the models written in the language, such as posterior inference and marginalisation. By providing these algorithms, a probabilistic programming language enables data scientists to focus on designing good models based on their domain knowledge, instead of building effective inference engines for their models, a task that typically requires expertise in machine learning, statistics and systems. Even experts in machine learning and statistics may benefit from such a probabilistic programming system because using the system they can easily explore highly advanced models. Several probabilistic programming languages have been built. Good examples are Stan [8], PyMC [23], Church [9], Venture [19], Anglican [35, 30], Turing [7], Pyro [3], Edward [32, 31], ProbTorch [26] and Hakaru [20].

In the past five years, with colleagues from programming languages, machine learning and probability theory, I have worked on developing the semantic foundations, efficient inference algorithms, and static program analysis for such probabilistic programming languages, especially those that support expressive language features such as higher-order functions, continuous distributions and general recursion. At FSCD'19, I plan to talk about some of these projects related to semantics and the lessons that I learnt.

This document is a companion to my FSCD'19 talk. Its primary goal is to provide keywords and references to the work mentioned in the talk, and help interested people start their follow-up study. If the reader looks for systematic introduction to probabilistic programming, I recommend to look at books [10, 34, 6] and teaching materials on probabilistic programming instead.

© Hongseok Yang;
licensed under Creative Commons License CC-BY

Listing 1 Anglican program with undefined posterior. (`normal_pdf x 0 1`) computes the density of the standard normal distribution at x. The program defines a model with two random variables x and y, the former being sampled from the standard normal distribution and the latter from the exponential distribution. The random variable y is observed to have the value 0, and the program expresses the posterior of x under this observation.

```
(let [x      (sample (normal 0 1))
      x_pdf  (normal_pdf x 0 1)
      y      (observe (exponential (/ 1 x_prob)) 0)]
  x)
```

2 Some research questions

A large part of my research has been about building a solid theoretical foundation for probabilistic programming languages. Doing so is particularly needed for such languages, because programs in those languages are run by inference engines but these inference engines only approximate the ideal mathematical semantics of these programs, namely, their posterior distributions. Even worse, sometimes probabilistic programs do not have posterior distributions at all, and I do not know of any inference engines that can detect it. Listing 1 shows one such program in Anglican, whose posterior distribution (or more precisely posterior density) is undefined.[1] Also, I found this foundation building intellectually rewarding, because, as I will explain shortly, it made me think about unexpected connections among multiple disciplines and revisit old concepts in programming languages, such as data abstraction, from a new perspective.

Here are three specific research questions about the foundation for probabilistic programming that intrigued me and my colleagues.

Q1: How to define a good denotational semantics for higher-order probabilistic programming languages with continuous distributions and general recursions?

A standard tool for defining a continuous probability distribution rigorously is measure theory. But it turns out that measure theory is not good enough for defining the denotational semantics of higher-order probabilistic programming languages, such as Church, Venture and Anglican, because measure theory does not support higher-order functions well. The category of measurable spaces is not Cartesian closed, because the set of measurable functions $[\mathbb{R} \to_m \mathbb{R}]$ cannot be turned into a measurable space that makes the following evaluation map measurable [1]:

$$ev : [\mathbb{R} \to_m \mathbb{R}] \times \mathbb{R} \to \mathbb{R}, \qquad ev(f, r) = f(r).$$

I have been involved in the joint efforts to address this semantic issue [29, 12, 25]. Using tools from category theory, we developed a theory of quasi-Borel spaces [12], which extends measure theory, and used our theory to define the denotational semantics of higher-order probabilistic programming languages and to prove the correctness of inference algorithms for

[1] Let $p_n(x, 0, 1)$ be the density at x of the normal distribution with mean 0 and standard deviation 1, and $p_\lambda(y)$ be the density of the exponential distribution with rate λ. The prior of the program in Listing 1 is $p_n(x, 0, 1)$, and the likelihood is the density of the $1/p_n(x, 0, 1)$-rate exponential distribution at $y = 0$, which is $p_{1/p_n(x,0,1)}(y = 0) = 1/p_n(x, 0, 1)$. Thus, the joint density is 1. The marginal likelihood is ∞. Thus, the posterior density $p(x|y = 0)$ is undefined.

Listing 2 Anglican program with non-differentiable density. The program denotes a model with three random variables x_1, x_2, y, all three being drawn from the normal distribution with different parameters. It expresses the posterior distribution of x_1 under the condition that y has the value 4.

```
(let [x1 (sample (normal 0 1))
      x2 (sample (normal (* x1 x1) 1))
      x3 (if (> x2 0) x1 x2)
      y  (observe (normal x3 1) 4)]
  x1)
```

such languages [25]. Recently, Vákár, Kammar and Staton built a domain theory on top of quasi-Borel spaces, and showed how to handle term and type recursions in denotational semantics in the presence of continuous probability distributions [33].

An interesting future direction is to generalise well-known results from probability theory using the theory of quasi-Borel spaces. Our initial investigation with de Finetti's theorem for exchangeable random sequences shows a promise [12].

Q2: Can a probabilistic program denote a distribution with a density that is not differentiable at some non-measure-zero set?

This question assumes a typical setting that gradient-based inference algorithms operate. In the setting, all sampling statements in probabilistic programs use distributions on $\mathbb{R}^n$ for some n that have densities with respect to the Lebesgue measure, and those probabilistic programs mean distributions on traces of sampled values during execution. For instance, the posterior distribution of the program in Listing 2 has the following density $f : \mathbb{R}^2 \to [0, \infty)$ with respect to the Lebesgue measure on $\mathbb{R}^2$: for $x_1, x_2 \in \mathbb{R}$,

$$f(x_1, x_2) = p_n(x_1, 0, 1) \cdot p_n(x_2, x_1^2, 1) \cdot (\mathbb{1}_{[x_2>0]} \cdot p_n(4, x_1, 1) + \mathbb{1}_{[x_2 \leq 0]} \cdot p_n(4, x_2, 1))$$

where $p_n(x, m, \sigma)$ is the density at x of the normal distribution with mean m and standard deviation σ and $\mathbb{1}_{[\varphi]}$ is the indicator function returning 1 if φ holds and 0 otherwise.

The negative answer to the question is needed in order for these gradient-based inference algorithms to work correctly [21, 18]. Intuitively, it ensures that the algorithms never attempt to compute gradients at non-differentiable points, and the effects of these non-differentiable points to the algorithms can be estimated algorithmically.

Currently we have only a partial answer to the question. We proved that for a first-order probabilistic programming language without loops, if a program in the language uses only analytic operations as its primitive operations, the set of its non-differentiable points has measure zero [36]. The question is open for a language that supports higher-order functions, includes loops, or permits non-analytic primitive operations.

Q3: What is a good theory of data abstraction for probabilistic programming languages, which in particular can let us analyse modules from Bayesian nonparametrics?

Sophisticated probabilistic models from Bayesian nonparametrics are implemented as modules in some probabilistic programming languages, such as Church and Anglican [24, 30]. This question asks for extending the theory of data abstraction to account for such modules. Ideally, the theory should provide guidance about how to implement such modules, and help programmers understand the consequences of using these modules.

I became interested in the question because of an intriguing feature of these modules. They are often implemented using impure features, such as mutable state, but they still satisfy a type of equations that typically hold for pure modules, such as commutativity of

module operations. It turns out that this phenomenon is not an accident; the probabilistic models that the modules denote satisfy symmetry properties such as exchangeability and contractibility [22, 16], and the equations for the modules come from these properties [28, 27].

Answering the question amounts to connecting the theory of data abstraction in programming languages to the study on these symmetry properties in probability theory. So far we found a connection for the Beta-Bernoulli process, one of the simplest models [27], and inspired by this connection, we defined a new type of symmetry properties for probabilistic models and proved a representation theorem for them [15]. These results are far from answering the question posed, and I expect (and hope) that more deep results are waiting to be discovered.

The three questions are chosen mainly based on my personal taste. If the reader wants to gain a broad view on what semantics researchers care about regarding probabilistic programming languages, I recommend to read the following papers [17, 14, 13, 4, 2, 5].

3 Final remark

Probabilistic programming is an exciting topic that raises several fresh theoretical and practical questions in programming languages, statistics, machine learning and probability theory. I hope that my FCSD'19 talk and (highly incomplete and biased) list of research questions in the previous section helps the reader partially understand why I and my colleagues are excited about the topic. This document conveys the view of one programming-language researcher on probabilistic programming. To understand how machine learning researchers think about probabilistic programming, I recommend to watch the video recording of Josh Tenenbaum's ICML'18 invited talk, and read the introduction of the book on probabilistic programming [34]; the introduction is written mostly by Frank Wood.

References

1. R. J. Aumann. Borel structures for function spaces. *Illinois Journal of Mathematics*, 5:614–630, 1961.
2. Sooraj Bhat, Johannes Borgström, Andrew D. Gordon, and Claudio V. Russo. Deriving Probability Density Functions from Probabilistic Functional Programs. *Logical Methods in Computer Science*, 13(2), 2017.
3. Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul A. Szerlip, Paul Horsfall, and Noah D. Goodman. Pyro: Deep Universal Probabilistic Programming. *Journal of Machine Learning Research*, 20:28:1–28:6, 2019.
4. J. Borgström, A. D. Gordon, M. Greenberg, J. Margetson, and J. van Gael. Measure Transformer Semantics for Bayesian Machine Learning. *LMCS*, 9(3):11, 2013.
5. Johannes Borgström, Ugo Dal Lago, Andrew D. Gordon, and Marcin Szymczak. A lambda-calculus foundation for universal probabilistic programming. In *ICFP*, pages 33–46, 2016.
6. Cameron Davidson-Pilon. *Bayesian Methods for Hackers: Probabilistic Programming and Bayesian Inference*. Addison-Wesley Professional, 2015.
7. Hong Ge, Kai Xu, and Zoubin Ghahramani. Turing: Composable inference for probabilistic programming. In *AISTATS*, pages 1682–1690, 2018.
8. Andrew Gelman, Daniel Lee, and Jiqiang Guo. Stan: A Probabilistic Programming Language for Bayesian Inference and Optimization. *Journal of Educational and Behavioral Statistics*, 40(5):530–543, 2015.
9. Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy, Keith Bonawitz, and Joshua B. Tenenbaum. Church: A Language for Generative Models. In *UAI*, pages 220–229, 2008.

10 Noah D Goodman and Andreas Stuhlmüller. The Design and Implementation of Probabilistic Programming Languages. `http://dippl.org`, 2014. Accessed: 2019-4-11.

11 Andrew D Gordon, Thomas A Henzinger, Aditya V Nori, and Sriram K Rajamani. Probabilistic programming. In *Proceedings of the on Future of Software Engineering, FOSE 2014*, pages 167–181, 2014.

12 Chris Heunen, Ohad Kammar, Sam Staton, and Hongseok Yang. A convenient category for higher-order probability theory. In *LICS*, pages 1–12, 2017.

13 Daniel Huang and Greg Morrisett. An Application of Computable Distributions to the Semantics of Probabilistic Programming Languages. In *ESOP*, pages 337–363, 2016.

14 Chung-Kil Hur, Aditya V. Nori, Sriram K. Rajamani, and Selva Samuel. A Provably Correct Sampler for Probabilistic Programs. In *FSTTCS*, pages 475–488, 2015.

15 Paul Jung, Jiho Lee, Sam Staton, and Hongseok Yang. A Generalization of Hierarchical Exchangeability on Trees to Directed Acyclic Graphs. *arXiv preprint*, 2018. `arXiv:1812.06282`.

16 Olav Kallenberg. *Probabilistic Symmetries and Invariance Principles.* Springer, 2005.

17 D. Kozen. Semantics of Probablistic Programs. *Journal of Computer and System Sciences*, 22:328–350, 1981.

18 Wonyeol Lee, Hangyeol Yu, and Hongseok Yang. Reparameterization Gradient for Non-differentiable Models. In *NeurIPS*, pages 5558–5568, 2018.

19 Vikash Mansinghka, Daniel Selsam, and Yura Perov. Venture: a higher-order probabilistic programming platform with programmable inference. *arXiv preprint*, 2014. `arXiv:1404.0099`.

20 Praveen Narayanan, Jacques Carette, Wren Romano, Chung-chieh Shan, and Robert Zinkov. Probabilistic inference by program transformation in Hakaru (system description). In *Proceedings of the 13th International Symposium on Functional and Logic Programming, FLOPS 2016*, pages 62–79, 2016.

21 Akihiko Nishimura, David Dunson, and Jianfeng Lu. Discontinuous Hamiltonian Monte Carlo for Sampling Discrete Parameters. *arXiv preprint*, 2017. `arXiv:1705.08510`.

22 Peter Orbanz and Daniel M. Roy. Bayesian Models of Graphs, Arrays and Other Exchangeable Random Structures. *IEEE Trans. Pattern Anal. Mach. Intell.*, 37(2):437–461, 2015.

23 Anand Patil, David Huard, and Christopher J Fonnesbeck. PyMC: Bayesian Stochastic Modelling in Python. *Journal of Statistical Software*, 35(4):1, 2010.

24 Daniel M. Roy, Vikash Mansinghka, Noah Goodman, and Joshua Tenenbaum. A stochastic programming perspective on nonparametric Bayes. In *ICML Workshop on Nonparametric Bayesian*, 2008.

25 Adam Scibior, Ohad Kammar, Matthijs Vákár, Sam Staton, Hongseok Yang, Yufei Cai, Klaus Ostermann, Sean K. Moss, Chris Heunen, and Zoubin Ghahramani. Denotational validation of higher-order Bayesian inference. *PACMPL*, 2(POPL):60:1–60:29, 2018.

26 N. Siddharth, Brooks Paige, Jan-Willem van de Meent, Alban Desmaison, Noah D. Goodman, Pushmeet Kohli, Frank Wood, and Philip Torr. Learning Disentangled Representations with Semi-Supervised Deep Generative Models. In *NIPS*, pages 5927–5937, 2017.

27 Sam Staton, Dario Stein, Hongseok Yang, Nathanael L. Ackerman, Cameron Freer, and Daniel M Roy. The Beta-Bernoulli Process and Algebraic Effects. In *ICALP*, 2018.

28 Sam Staton, Hongseok Yang, Nathanael L. Ackerman, Cameron Freer, and Daniel M Roy. Exchangeable random process and data abstraction. In *Workshop on Probabilistic Programming Semantics, PPS 2017*, 2017.

29 Sam Staton, Hongseok Yang, Chris Heunen, Ohad Kammar, and Frank Wood. Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints. In *LICS*, pages 525–534, 2016.

30 David Tolpin, Jan-Willem van de Meent, Hongseok Yang, and Frank D. Wood. Design and Implementation of Probabilistic Programming Language Anglican. In *Proceedings of the 28th Symposium on the Implementation and Application of Functional Programming Languages, IFL 2016*, pages 6:1–6:12, 2016.

31 Dustin Tran, Matthew D. Hoffman, Dave Moore, Christopher Suter, Srinivas Vasudevan, and Alexey Radul. Simple, Distributed, and Accelerated Probabilistic Programming. In *NeurIPS*, pages 7609–7620, 2018.

32 Dustin Tran, Alp Kucukelbir, Adji B. Dieng, Maja R. Rudolph, Dawen Liang, and David M. Blei. Edward: A library for probabilistic modeling, inference, and criticism. *CoRR*, abs/1610.09787, 2016.

33 Matthijs Vákár, Ohad Kammar, and Sam Staton. A domain theory for statistical probabilistic programming. *PACMPL*, 3(POPL):36:1–36:29, 2019.

34 Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. An Introduction to Probabilistic Programming. *arXiv preprint*, 2018. `arXiv:1809.10756`.

35 Frank Wood, Jan Willem Meent, and Vikash Mansinghka. A New Approach to Probabilistic Programming Inference. In *AISTATS*, pages 1024–1032, 2014.

36 Yuan Zhou, Bradley Gram-Hansen, Tobias Kohn, Tom Rainforth, Hongseok Yang, and Frank Wood. A Low-Level Probabilistic Programming Language for Non-Differentiable Models. In *AISTATS*, 2019.

Bicategories in Univalent Foundations

Benedikt Ahrens
School of Computer Science, University of Birmingham, United Kingdom
b.ahrens@cs.bham.ac.uk

Dan Frumin
Institute for Computation and Information Sciences,
Radboud University, Nijmegen, The Netherlands
dfrumin@cs.ru.nl

Marco Maggesi
Dipartimento di Matematica e Informatica "Dini", Università degli Studi di Firenze, Italy
marco.maggesi@unifi.it

Niels van der Weide
Institute for Computation and Information Sciences,
Radboud University, Nijmegen, The Netherlands
nweide@cs.ru.nl

Abstract

We develop bicategory theory in univalent foundations. Guided by the notion of univalence for (1-)categories studied by Ahrens, Kapulkin, and Shulman, we define and study univalent bicategories. To construct examples of those, we develop the notion of "displayed bicategories", an analog of displayed 1-categories introduced by Ahrens and Lumsdaine. Displayed bicategories allow us to construct univalent bicategories in a modular fashion. To demonstrate the applicability of this notion, we prove several bicategories are univalent. Among these are the bicategory of univalent categories with families and the bicategory of pseudofunctors between univalent bicategories. Our work is formalized in the `UniMath` library of univalent mathematics.

2012 ACM Subject Classification Theory of computation → Type theory

Keywords and phrases bicategory theory, univalent mathematics, dependent type theory, Coq

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.5

Funding Work on this article was supported by a grant from the COST Action EUTypes CA15123.
Benedikt Ahrens: This material is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-17-1-0363. Ahrens acknowledges the support of the Centre for Advanced Study (CAS) in Oslo, Norway, which funded and hosted the research project *Homotopy Type Theory and Univalent Foundations* during the 2018/19 academic year.
Dan Frumin: Supported by the Netherlands Organisation for Scientific Research (NWO/TTW) under the STW project 14319.
Marco Maggesi: MIUR, GNSAGA-INdAM.

Acknowledgements We would like to express our gratitude to all the EUTypes actors for their support. We also thank Niccolò Veltri for commenting on a draft of this paper. Finally, we thank the referees for their careful reading and thoughtful and constructive criticism.

© Benedikt Ahrens, Dan Frumin, Marco Maggesi, and Niels van der Weide;
licensed under Creative Commons License CC-BY

1 Introduction

Category theory (by which we mean 1-category theory) is established as a convenient language to structure, and discuss, mathematical objects and maps between them. To axiomatize the fundamental objects of category theory itself – categories, functors, and natural transformations – the theory of 1-categories is not enough. Instead, category-like structures allowing for "morphisms between morphisms" were developed to account for the natural transformations. Among those are *bicategories*.

Bicategory theory was originally developed by Bénabou [5] in set-theoretic foundations. The goal of our work is to develop bicategory theory in univalent foundations. Specifically, we give here a notion of *univalent bicategory* and show that some bicategories of interest are univalent. To this end, we generalize *displayed categories* [2] to the bicategorical setting, and prove that the total bicategory spanned by a displayed bicategory is univalent, if the constituent pieces are.

Univalent foundations and categories therein. According to Voevodsky, a foundation of mathematics consists of three things:

1. a language for mathematical objects;
2. a notion of proposition and proof; and
3. an interpretation of those into a world of mathematical objects.

By "univalent foundations", we mean the foundation given by univalent type theory as described, *e.g.,* in the HoTT book [21], with its notion of "univalent logic", and the interpretation of univalent type theory in simplicial sets expected to arise from Voevodsky's simplicial set model [14].

In the simplicial set model, univalent categories (just called "categories" in [1]) correspond to truncated complete Segal spaces, which in turn are equivalent to ordinary (set-theoretic) categories. This means that univalent categories are "the right" notion of categories in univalent foundations: they correspond exactly to the traditional set-theoretic notion of category. Similarly, the notion of *univalent bicategory*, studied in this paper, provides the correct notion of bicategory in univalent foundations.

Throughout this article, we work in type theory with function extensionality. We explicitly mention any use of the univalence axiom. We use the notation standardized in [21]; a significantly shorter overview of the setting we work in is given in [1]. As a reference for 1-category theory in univalent foundations, we refer to [1], which follows a path suggested by Hofmann and Streicher [13, Section 5.5].

Bicategories for Type Theory. Our motivation for this work stems from several particular (classes of) bicategories, that come up in our work on the semantics of type theories and Higher Inductive Types (HITs).

Firstly, we are interested in the "categories with structure" that have been used in the model theory of type theories. The purpose of the various categorical structures is to model context extension and substitution. Prominent such notions are categories with families (see, *e.g.,* [8, 9]), categories with attributes (see, *e.g.,* [19]), and categories with display maps (see, *e.g.,* [20]). Each notion of "categorical structure" gives rise to a bicategory whose objects are categories equipped with such a structure. Secondly, in the study of HITs, bicategories of algebras feature prominently, see, *e.g.,* work by Dybjer and Moenclaey [10]. Our long term goal is to show that these bicategories are univalent.

Displayed bicategories. In this work, we develop the notion of *displayed bicategory* analogous to the 1-categorical notion of displayed category introduced in [2]. Intuitively, a displayed bicategory D over a bicategory B represents data and properties to be added to B to form a new bicategory: D gives rise to the *total bicategory* $\int \mathsf{D}$. Its cells are pairs (b, d) where d in D is a "displayed cell" over b in B.

When a bicategory is built as the total bicategory $\int \mathsf{D}$ of a displayed bicategory D over base B, univalence of $\int \mathsf{D}$ can be shown from univalence of B and "displayed univalence" of D. The latter two conditions are easier to show, sometimes significantly easier.

Two features make the displayed point of view particularly useful: firstly, displayed structures can be *iterated*, making it possible to build bicategories of very complicated objects layerwise. Secondly, displayed "building blocks" can be provided, for which univalence is proved once and for all. These building blocks, *e.g.,* cartesian product, can be used like LEGO™ pieces to *modularly* build complicated bicategories that are automatically accompanied by a proof of univalence.

We demonstrate these features in examples, proving univalence of three complicated (classes of) bicategories: first, the bicategory of pseudofunctors between two univalent bicategories; second, bicategories of algebraic structures; and third, the bicategory of categories with families.

Formalization. The results presented here are mechanized in the `UniMath` library [22], which is based on the Coq proof assistant [17]. The `UniMath` library is under constant development; in this paper, we refer to the version with `git` hash ab97d96. Throughout the paper, definitions and statements are accompanied by a link to the online documentation of that version. For instance, the link `bicat` points to the definition of a bicategory.

Related work. Our work extends the notion of univalence from 1-categories [1] to bicategories. Similarly, we extend the notion of displayed 1-category [2] to the bicategorical setting.

Capriotti and Kraus [7] study univalent $(n, 1)$-categories for $n \in \{0, 1, 2\}$. They only consider bicategories where the 2-cells are equalities between 1-cells; in particular, all 2-cells in [7] are invertible, and their $(2, 1)$-categories are by definition locally univalent. Consequently, the condition called *univalence* by Capriotti and Kraus is what we call *global univalence*, cf. Definition 3.1, Item 2. In this work, we study bicategories, a.k.a. $(2, 2)$-categories, that is, we allow for non-invertible 2-cells. The examples we study in Section 6 are proper $(2, 2)$-categories and are not covered by [7].

Lafont, Hirschowitz, and Tabareau [15] are working on formalizing ω-categories in type theory. Their work is guided by work by Finster and Mimram [11], who develop a type theory for which the models (in set theory) are precisely weak ω-categories.

2 Bicategories and Some Examples

Bicategories were introduced by Bénabou [5] in 1967, encompassing monoidal categories, 2-categories (in particular, the 2-category of categories), and other examples. He (and later many other authors) defines bicategories in the style of "categories weakly enriched in categories". That is, the homs $\mathsf{B}_1(a, b)$ of a bicategory B are taken to be (1-)categories, and composition is given by a functor $\mathsf{B}_1(a, b) \times \mathsf{B}_1(b, c) \to \mathsf{B}_1(a, c)$. This presentation of bicategories is concise and convenient for communication between mathematicians.

In this article, we use a different, more unfolded definition of bicategories, which is inspired by Bénabou [5, Section 1.3] and [18, Section "Details"]. It is more verbose than the definition via weak enrichment. However, it is better suited for our purposes, in particular, it is suitable for defining *displayed bicategories*, cf. Section 4.

▶ **Definition 2.1** (`bicat`). A **bicategory** B consists of

1. of a type B_0 of **objects**;
2. a type $\mathsf{B}_1(a,b)$ of **1-cells** for all $a, b : \mathsf{B}_0$;
3. a ***set*** $\mathsf{B}_2(f,g)$ of **2-cells** for all $a, b : \mathsf{B}_0$ and $f, g : \mathsf{B}_1(a,b)$;
4. an **identity 1-cell** $\mathrm{id}_1(a) : \mathsf{B}_1(a,a)$;
5. a **composition** $\mathsf{B}_1(a,b) \times \mathsf{B}_1(b,c) \to \mathsf{B}_1(a,c)$, written $f \cdot g$;
6. an **identity 2-cell** $\mathrm{id}_2(f) : \mathsf{B}_2(f,f)$;
7. a **vertical composition** $\theta \bullet \gamma : \mathsf{B}_2(f,h)$ for all 1-cells $f, g, h : \mathsf{B}_1(a,b)$ and 2-cells $\theta : \mathsf{B}_2(f,g)$ and $\gamma : \mathsf{B}_2(g,h)$;
8. a **left whiskering** $f \lhd \theta : \mathsf{B}_2(f \cdot g, f \cdot h)$ for all 1-cells $f : \mathsf{B}_1(a,b)$ and $g, h : \mathsf{B}_1(b,c)$ and 2-cells $\theta : \mathsf{B}_2(g,h)$;
9. a **right whiskering** $\theta \rhd h : \mathsf{B}_2(f \cdot h, g \cdot h)$ for all 1-cells $f, g : \mathsf{B}_1(a,b)$ and $h : \mathsf{B}_1(b,c)$ and 2-cells $\theta : \mathsf{B}_2(f,g)$;
10. a **left unitor** $\lambda(f) : \mathsf{B}_2(\mathrm{id}_1(a) \cdot f, f)$ and its inverse $\lambda(f)^{-1} : \mathsf{B}_2(f, \mathrm{id}_1(a) \cdot f)$;
11. a **right unitor** $\rho(f) : \mathsf{B}_2(f \cdot \mathrm{id}_1(b), f)$ and its inverse $\rho(f)^{-1} : \mathsf{B}_2(f, f \cdot \mathrm{id}_1(b))$;
12. a **left associator** $\alpha(f,g,h) : \mathsf{B}_2(f \cdot (g \cdot h), (f \cdot g) \cdot h)$ and a right associator $\alpha(f,g,h)^{-1} : \mathsf{B}_2((f \cdot g) \cdot h, f \cdot (g \cdot h))$ for $f : \mathsf{B}_1(a,b)$, $g : \mathsf{B}_1(b,c)$, and $h : \mathsf{B}_1(c,d)$

such that, for all suitable objects, 1-cells, and 2-cells,

13. $\mathrm{id}_2(f) \bullet \theta = \theta, \quad \theta \bullet \mathrm{id}_2(g) = \theta, \quad \theta \bullet (\gamma \bullet \tau) = (\theta \bullet \gamma) \bullet \tau$;
14. $f \lhd (\mathrm{id}_2\, g) = \mathrm{id}_2(f \cdot g), \quad f \lhd (\theta \bullet \gamma) = (f \lhd \theta) \bullet (f \lhd \gamma)$;
15. $(\mathrm{id}_2\, f) \rhd g = \mathrm{id}_2(f \cdot g), \quad (\theta \bullet \gamma) \rhd g = (\theta \rhd g) \bullet (\gamma \rhd g)$;
16. $(\mathrm{id}_1(a) \lhd \theta) \bullet \lambda(g) = \lambda(f) \bullet \theta$;
17. $(\theta \rhd \mathrm{id}_1(b)) \bullet \rho(g) = \rho(f) \bullet \theta$;
18. $(f \lhd (g \lhd \theta)) \bullet \alpha(f,g,i) = \alpha(f,g,h) \bullet ((f \cdot g) \lhd \theta)$;
19. $(f \lhd (\theta \rhd i)) \bullet \alpha(f,h,i) = \alpha(f,g,i) \bullet ((f \lhd \theta) \rhd i)$;
20. $(\theta \rhd (h \cdot i)) \bullet \alpha(g,h,i) = \alpha(f,h,i) \bullet ((\theta \rhd h) \rhd i)$;
21. $\lambda(f) \bullet \lambda(f)^{-1} = \mathrm{id}_2(\mathrm{id}_1(a) \cdot f), \quad \lambda(f)^{-1} \bullet \lambda(f) = \mathrm{id}_2(f)$;
22. $\rho(f) \bullet \rho(f)^{-1} = \mathrm{id}_2(f \cdot \mathrm{id}_1(b)), \quad \rho(f)^{-1} \bullet \rho(f) = \mathrm{id}_2(f)$;
23. $\alpha(f,g,h) \bullet \alpha(f,g,h)^{-1} = \mathrm{id}_2(f \cdot (g \cdot h)), \quad \alpha(f,g,h)^{-1} \bullet \alpha(f,g,h) = \mathrm{id}_2((f \cdot g) \cdot h)$;
24. $\alpha(f, \mathrm{id}_1(b), g) \bullet (\rho(f) \rhd g) = f \lhd \lambda(f)$;
25. $\alpha(f, g, h \cdot i) \bullet \alpha(f \cdot g, h, i) = (f \lhd \alpha(g,h,i)) \bullet \alpha(f, g \cdot h, i) \bullet (\alpha(f,g,h) \rhd i)$.

We write $a \to b$ for $\mathsf{B}_1(a,b)$ and $f \Rightarrow g$ for $\mathsf{B}_2(f,g)$. Riley formalized a definition of bicategories via weak enrichment in `UniMath`, based on work by Lumsdaine. These two definitions are equivalent.

▶ **Proposition 2.2.** *The definition of bicategories given in Definition 2.1 is equivalent to the formalized definition in terms of weak enrichment.*

This result is not formalized in our computer-checked library. However, as a sanity check for our definition of bicategory, we constructed maps between the two variations of bicategories, see `BicategoryOfBicat.v` and `BicatOfBicategory.v`.

Recall that our goal is to study univalence of bicategories, which is a property that relates equivalence and equality. For this reason, we study the two analogs of the 1-categorical notion of isomorphism. The first one is the notion of *invertible 2-cells*.

▶ **Definition 2.3** (`is_invertible_2cell`). A 2-cell $\theta : f \Rightarrow g$ is called **invertible** if we have $\gamma : g \Rightarrow f$ such that $\theta \bullet \gamma = \mathrm{id}_2(f)$ and $\gamma \bullet \theta = \mathrm{id}_2(g)$. An **invertible 2-cell** consists of a 2-cell and a proof that it is invertible, and $\mathsf{inv2cell}(f,g)$ is the type of invertible 2-cells from f to g.

Since inverses are unique, being an invertible 2-cell is a proposition. In addition, $\mathrm{id}_2(f)$ is invertible, and we write $\mathrm{id}_2(f) : \mathsf{inv2cell}(f, f)$ for this invertible 2-cell. The second analog of isomorphisms is the notion of *adjoint equivalences*.

▶ **Definition 2.4** (`adjoint_equivalence`)**.** An **adjoint equivalence structure** on a 1-cell $f : a \to b$ consists of a 1-cell $g : b \to a$ and invertible 2-cells $\eta : \mathrm{id}_2(f) \Rightarrow f \cdot g$ and $\varepsilon : g \cdot f \Rightarrow \mathrm{id}_2(g)$ together with paths

$$\rho(g)^{-1} \bullet (g \triangleleft \eta) \bullet \alpha(g, f, g)^{-1} \bullet (\varepsilon \triangleright g) \bullet \lambda(g) = \mathrm{id}_2(g),$$
$$\lambda(g)^{-1} \bullet (\eta \triangleright f) \bullet \alpha(f, g, f) \bullet (f \triangleleft \varepsilon) \bullet \rho(f) = \mathrm{id}_2(f).$$

An **adjoint equivalence** consists of a map f together with an adjoint equivalence structure on f. The type $\mathsf{AdjEquiv}(a, b)$ consists of all adjoint equivalences from a to b.

We call η and ε the unit and counit of the adjunction, and we call g the right adjoint. The prime example of an adjoint equivalence is the identity 1-cell $\mathrm{id}_1(a)$ and we denote it by $\mathrm{id}_1(a) : \mathsf{AdjEquiv}(a, a)$. Sometimes, we write $a \simeq b$ for $\mathsf{AdjEquiv}(a, b)$.

Before we continue our study of univalence, we present some examples of bicategories.

▶ **Example 2.5** (`fundamental_bigroupoid`)**.** Let X be a 2-type. Then we define a bicategory whose 0-cells are inhabitants of X, 1-cells from x to y are paths $x = y$, and 2-cells from p to q are higher paths $p = q$. The operations are defined with path induction. Every 1-cell is an adjoint equivalence and every 2-cell is invertible.

▶ **Example 2.6** (`one_types`)**.** Let U be a universe. The objects of the bicategory $1\text{-}\mathsf{Type}_{\mathsf{U}}$ of 1-types from U are 1-truncated types of the universe U, the 1-cells are functions between the underlying types, and the 2-cells are homotopies between functions. The 1-cells $\mathrm{id}_1(X)$ and $f \cdot g$ are defined as the identity and composition of functions. The 2-cell $\mathrm{id}_2(f)$ is refl, the 2-cell $p \bullet q$ is the concatenation of paths. The unitors and associators are defined as the identity path. Every 2-cell is invertible and adjoint equivalences between X and Y are the same as equivalences from X to Y.

▶ **Example 2.7** (`bicat_of_cats`)**.** We define the bicategory Cat of univalent categories as the bicategory whose 0-cells are univalent categories, 1-cells are functors, and 2-cells are natural transformations. For the operations, we use the identity and composition of functors, and whiskering of functors and transformations. The internal invertible 2-cells are the natural isomorphisms of functors, and the internal adjoint equivalences correspond to external adjoint equivalences of categories.

3 Univalent Bicategories

Recall that a (1-)category C (called "precategory" in [1]) is called *univalent* if, for every two objects $a, b : \mathsf{C}_0$, the canonical map $\mathsf{idtoiso}_{a,b} : (a = b) \to \mathsf{Iso}(a, b)$ from identities between a and b to isomorphisms between them is an equivalence. For bicategories, where we have one more layer of structure, univalence can be imposed both *locally* and *globally*.

▶ **Definition 3.1** (`Univalence.v`)**.** Univalence for bicategories is defined as follows:

1. Let $a, b : \mathsf{B}_0$ and $f, g : \mathsf{B}_1(a, b)$ be objects and morphisms of B; by path induction we define a map $\mathsf{idtoiso}^{2,1}_{f,g} : f = g \to \mathsf{inv2cell}(f, g)$ which sends $\mathsf{refl}(f)$ to $\mathrm{id}_2(f)$. A bicategory B is **locally univalent** if, for every two objects $a, b : \mathsf{B}_0$ and two 1-cells $f, g : \mathsf{B}_1(a, b)$, the map $\mathsf{idtoiso}^{2,1}_{f,g}$ is an equivalence.

2. Let $a, b : \mathsf{B}_0$ be objects of B; using path induction we define $\mathsf{idtoiso}^{2,0}_{a,b} : a = b \to \mathsf{AdjEquiv}(a, b)$ sending $\mathsf{refl}(a)$ to $\mathrm{id}_1(a)$. A bicategory B is **globally univalent** if, for every two objects $a, b : \mathsf{B}_0$, the canonical map $\mathsf{idtoiso}^{2,0}_{a,b}$ is an equivalence.
3. (`is_univalent_2`) We say that B is **univalent** if B is both locally and globally univalent.

While right adjoints are only unique up to equivalence in general, they are unique up to identity when the bicategory is locally univalent:

▶ **Proposition 3.2** (`isaprop_left_adjoint_equivalence`). *Let* B *be locally univalent. Then having an adjoint equivalence structure on a 1-cell in* B *is a proposition.*

As a corollary of this proposition we get the following:

▶ **Theorem 3.3.** *In a univalent bicategory* B*, the type* B_0 *of 0-cells is a 2-type, and for any two objects* $a, b : \mathsf{B}_0$*, the type* $a \to b$ *of 1-cells from* a *to* b *is a 1-type.*

To prove global univalence of a bicategory, we need to show that $\mathsf{idtoiso}^{2,0}_{a,b}$ is an equivalence. Often we do that by constructing a map in the other direction and showing these two are inverses. This requires comparing adjoint equivalences, which is done with the help of Proposition 3.2.

Now let us prove the examples from Section 2 are univalent.

▶ **Example 3.4.** The following bicategories are univalent:
1. (`TwoType.v`, Example 2.5 cont'd) The fundamental bigroupoid of each 2-type is univalent.
2. (`OneTypes.v`, Example 2.6 cont'd) The bicategory of 1-types of a universe U is locally univalent; this is a consequence of function extensionality. If we assume the univalence axiom for U, then 1-types form a univalent bicategory.

It is more difficult to prove the bicategory of univalent categories is univalent, and we only give a brief sketch of this proof.

▶ **Proposition 3.5** (`BicatOfCats.v`, Example 2.7 cont'd). *The bicategory* Cat *is univalent.*

Local univalence follows from the fact that the functor category $[C, D]$ is univalent if D is. For global univalence, we use that the type of identities on categories is equivalent to the type of adjoint equivalences between categories [1, Theorem 6.17]. The proof proceeds by factoring $\mathsf{idtoiso}^{2,0}$ as a chain of equivalences $(C = D) \xrightarrow{\sim} \mathsf{CatIso}(C, D) \xrightarrow{\sim} \mathsf{AdjEquiv}(C, D)$. To our knowledge, a proof of global univalence was first computer-formalized by Rafaël Bocquet[1].

In the previous examples, we proved univalence directly. However, in many complicated bicategories such proofs are not feasible. An example of such a bicategory is the bicategory $\mathsf{Pseudo}(\mathsf{B}, \mathsf{C})$ of pseudofunctors from B to C, pseudotransformations, and modifications [16] (for a univalent bicategory C). Even in the 1-categorical case, proving the univalence of the category $[C, D]$ of functors from C to D, and natural transformations between them, is tedious. In Section 5, we develop some machinery to prove the following theorem.

▶ **Theorem 3.6** (`psfunctor_bicat_is_univalent_2`). *If* B *is a (not necessarily univalent) bicategory and* C *is a univalent bicategory, then the bicategory* $\mathsf{Pseudo}(\mathsf{B}, \mathsf{C})$ *of pseudofunctors from* B *to* C *is univalent.*

[1] `https://github.com/mortberg/cubicaltt/blob/master/examples/category.ctt`

4 Displayed Bicategories

In this section, we introduce *displayed bicategories*, the bicategorical analog to the notion of displayed category developed in [2]. A displayed (1-)category D over a given (base) category C consists of a family of objects over objects in C and a family of morphisms over morphisms in C together with suitable displayed operations of composition and identity. A category $\int D$ is then constructed, the objects and morphisms of which are pairs of objects and morphisms from C and D, respectively. Properties of $\int D$, in particular univalence, can be shown from analogous, but simpler, conditions on C and D.

A prototypical example is the following displayed category over $C := \mathsf{Set}$: an object over a set X is a group structure on X, and a morphism over a function $f : X \to X'$ from group structure G (on X) to group structure G' (on X') is a proof of the fact that f is compatible with G and G'. The total category is the category of groups, and its univalence follows from univalence of Set and a univalence property of the displayed data.

Just like in 1-category theory, many examples of bicategories are obtained by endowing previously considered bicategories with additional structure. An example is the bicategory of pointed 1-types in U. The objects in this bicategory are pairs of a 1-type A and an inhabitant $a : A$. The morphisms are pairs of a morphism f of 1-types and a path witnessing that f preserves the selected points. Similarly, the 2-cells are pairs of a homotopy p and a proof that this p commutes with the point preservation proofs. Thus, this bicategory is obtained from $1\text{-}\mathsf{Type}_{\mathsf{U}}$ by endowing the cells on each level with additional structure.

Of course, the structure should be added in such a way that we are guaranteed to obtain a bicategory at the end. Now let us give the formal definition of displayed bicategories.

▶ **Definition 4.1** (`disp_bicat`)**.** Given a bicategory B, a **displayed bicategory** D **over** B is given by data analogous to that of a bicategory, to which the numbering refers:

1. for each $a : \mathsf{B}_0$ a type D_a of displayed 0-cells over a;
2. for each $f : a \to b$ in B and $\bar{a} : \mathsf{D}_a, \bar{b} : \mathsf{D}_b$ a type $\bar{a} \xrightarrow{f} \bar{b}$ of displayed 1-cells over f;
3. for each $\theta : f \Rightarrow g$ in B, $\bar{f} : \bar{a} \xrightarrow{f} \bar{b}$ and $\bar{g} : \bar{a} \xrightarrow{g} \bar{b}$ a set $\bar{f} \overset{\theta}{\Rightarrow} \bar{g}$ of displayed 2-cells over θ

and dependent versions of operations and laws from Definition 2.1, which are

4. for each $a : \mathsf{B}_0$ and $\bar{a} : \mathsf{D}_0(a)$, we have $\mathrm{id}_1(\bar{a}) : \bar{a} \xrightarrow{\mathrm{id}_1(a)} \bar{a}$;
5. for all 1-cells $f : a \to b$, $g : b \to c$, and displayed 1-cells $\bar{f} : \bar{a} \xrightarrow{f} \bar{b}$ and $\bar{g} : \bar{b} \xrightarrow{g} \bar{c}$, we have a displayed 1-cell $\bar{f} \cdot \bar{g} : \bar{a} \xrightarrow{f \cdot g} \bar{c}$;
6. for all $f : \mathsf{B}_1(a, b)$, $\bar{a} : \mathsf{D}_0(a)$, $\bar{b} : \mathsf{D}_0(b)$, and $\bar{f} : \bar{a} \xrightarrow{f} \bar{b}$, we have $\mathrm{id}_2(\bar{f}) : \bar{f} \xRightarrow{\mathrm{id}_2(f)} \bar{f}$;
7. for 2-cells $\theta : f \Rightarrow g$ and $\gamma : g \Rightarrow h$, and displayed 2-cells $\bar{\theta} : \bar{f} \overset{\theta}{\Rightarrow} \bar{g}$ and $\bar{\gamma} : \bar{g} \xrightarrow{\gamma} \bar{h}$, we have a displayed 2-cell $\bar{\theta} \bullet \bar{\gamma} : \bar{f} \xRightarrow{\theta \bullet \gamma} \bar{h}$.
8. for each displayed 1-cell $\bar{f} : \bar{a} \xrightarrow{f} \bar{b}$ and each displayed 2-cell $\bar{g} \overset{\theta}{\Rightarrow} \bar{h}$, we have a displayed 2-cell $\bar{f} \triangleleft \bar{\theta} : \bar{f} \cdot \bar{g} \xRightarrow{f \triangleleft \theta} \bar{f} \cdot \bar{h}$;
9. for each displayed 1-cell $\bar{h} : \bar{b} \xrightarrow{h} \bar{c}$ and each displayed 2-cell $\bar{\theta} : \bar{f} \overset{\theta}{\Rightarrow} \bar{g}$, we have a displayed 2-cell $\bar{\theta} \triangleright \bar{h} : \bar{f} \cdot \bar{h} \xRightarrow{\theta \triangleright h} \bar{g} \cdot \bar{h}$;
10. for each $\bar{f} : \bar{a} \xrightarrow{f} \bar{b}$, we have displayed 2-cells $\lambda(\bar{f}) : \mathrm{id}_1(\bar{a}) \cdot \bar{f} \overset{\lambda f}{\Longrightarrow} \bar{f}$ and $\lambda(\bar{f})^{-1} : \bar{f} \xRightarrow{\lambda f^{-1}} \mathrm{id}_1(\bar{a}) \cdot \bar{f}$;
11. for each $\bar{f} : \bar{a} \xrightarrow{f} \bar{b}$, displayed 2-cells $\rho(\bar{f}) : \bar{f} \cdot \mathrm{id}_1(\bar{b}) \overset{\rho f}{\Longrightarrow} \bar{f}$ and $\rho(\bar{f})^{-1} : \bar{f} \xRightarrow{\rho f^{-1}} \bar{f} \cdot \mathrm{id}_1(\bar{b})$;
12. for each $\bar{f} : \bar{a} \overset{f}{\Rightarrow} \bar{b}$, $\bar{g} : \bar{b} \overset{g}{\Rightarrow} \bar{c}$, and $\bar{h} : \bar{c} \overset{h}{\Rightarrow} \bar{d}$, we have displayed 2-cells $\alpha(\bar{f}, \bar{g}, \bar{h}) : \bar{f} \cdot (\bar{g} \cdot \bar{h}) \xRightarrow{\alpha(f,g,h)} (\bar{f} \cdot \bar{g}) \cdot \bar{h}$ and $\alpha(\bar{f}, \bar{g}, \bar{h})^{-}1 : (\bar{f} \cdot \bar{g}) \cdot \bar{h} \xRightarrow{\alpha(f,g,h)} \bar{f} \cdot (\bar{g} \cdot \bar{h})$.

Note that we use the same notation for the displayed and the non-displayed operations.

These operations are subject to laws, which are derived systematically from the non-displayed version. Just as for displayed 1-categories, the laws of displayed bicategories are heterogeneous, because they are transported along the analogous law in the base bicategory. For instance, the displayed left-unitary law for identity reads as $\mathrm{id}_2(\bar{f}) \bullet \bar{\theta} =_e \bar{\theta}$, where e is the corresponding identity of Item 13 in Definition 2.1.

13. $\mathrm{id}_2(f) \bullet \theta =_* \theta, \quad \theta \bullet \mathrm{id}_2(g) =_* \theta, \quad \theta \bullet (\gamma \bullet \tau) =_* (\theta \bullet \gamma) \bullet \tau;$
14. $f \lhd (\mathrm{id}_2\, g) =_* \mathrm{id}_2(f \cdot g), \quad f \lhd (\theta \bullet \gamma) =_* (f \lhd \theta) \bullet (f \lhd \gamma);$
15. $(\mathrm{id}_2\, f) \rhd g =_* \mathrm{id}_2(f \cdot g), \quad (\theta \bullet \gamma) \rhd g =_* (\theta \rhd g) \bullet (\gamma \rhd g);$
16. $(\mathrm{id}_1(a) \lhd \theta) \bullet \lambda(g) =_* \lambda(f) \bullet \theta;$
17. $(\theta \rhd \mathrm{id}_1(b)) \bullet \rho(g) =_* \rho(f) \bullet \theta;$
18. $(f \lhd (g \lhd \theta)) \bullet \alpha(f, g, i) =_* \alpha(f, g, h) \bullet ((f \cdot g) \lhd \theta);$
19. $(f \lhd (\theta \rhd i)) \bullet \alpha(f, h, i) =_* \alpha(f, g, i) \bullet ((f \lhd \theta) \rhd i);$
20. $(\theta \rhd (h \cdot i)) \bullet \alpha(g, h, i) =_* \alpha(f, h, i) \bullet ((\theta \rhd h) \rhd i);$
21. $\lambda(f) \bullet \lambda(f)^{-1} =_* \mathrm{id}_2(\mathrm{id}_1(a) \cdot f), \quad \lambda(f)^{-1} \bullet \lambda(f) =_* \mathrm{id}_2(f);$
22. $\rho(f) \bullet \rho(f)^{-1} =_* \mathrm{id}_2(f \cdot \mathrm{id}_1(b)), \quad \rho(f)^{-1} \bullet \rho(f) =_* \mathrm{id}_2(f);$
23. $\alpha(f, g, h) \bullet \alpha(f, g, h)^{-1} =_* \mathrm{id}_2(f \cdot (g \cdot h)), \quad \alpha(f, g, h)^{-1} \bullet \alpha(f, g, h) =_* \mathrm{id}_2((f \cdot g) \cdot h);$
24. $\alpha(f, \mathrm{id}_1(b), g) \bullet (\rho(f) \rhd g) =_* f \lhd \lambda(f);$
25. $\alpha(f, g, h \cdot i) \bullet \alpha(f \cdot g, h, i) =_* (f \lhd \alpha(g, h, i)) \bullet \alpha(f, g \cdot h, i) \bullet (\alpha(f, g, h) \rhd i).$

The purpose of displayed bicategories is to give rise to a total bicategory together with a projection pseudofunctor. They are defined as follows:

▶ **Definition 4.2** (`total_bicat`). Given a displayed bicategory D over a bicategory B, we can form a **total bicategory** $\int \mathsf{D}$ (or $\int_{\mathsf{B}} \mathsf{D}$) which has:

1. as 0-cells tuples $(a, \bar{a})$, where $a : \mathsf{B}$ and $\bar{a} : \mathsf{D}_a$;
2. as 1-cells tuples $(f, \bar{f}) : (a, \bar{a}) \to (b, \bar{b})$, where $f : a \to b$ and $\bar{f} : \bar{a} \xrightarrow{f} \bar{b}$;
3. as 2-cells tuples $(\theta, \bar{\theta}) : (f, \bar{f}) \Rightarrow (g, \bar{g})$, where $\theta : f \Rightarrow g$ and $\bar{\theta} : \bar{f} \overset{\theta}{\Rightarrow} \bar{g}$.

We also have a **projection pseudofunctor** $\pi_D : \mathsf{Pseudo}(\int D, \mathsf{B})$.

As mentioned before, the bicategory of pointed 1-types is the total bicategory of the following displayed bicategory.

▶ **Example 4.3** (`p1types_disp`, Example 3.4, Item 2 cont'd). Given a universe U, we build a displayed bicategory of pointed 1-types over the base bicategory of 1-types in U (Example 2.6).

- For 1-type A in U, the objects over A are inhabitants of A.
- For $f : A \to B$ with A, B 1-types in U, the maps over f from a to b are paths $f(a) = b$.
- Given two maps $f, g : A \to B$, a path $p : f = g$, two points $a : A$ and $b : B$, and paths $q_f : f(a) = b$ and $q_g : g(a) = b$, the 2-cells over p are paths $\mathsf{transport}^{x \mapsto x = b}(p, q_f) = q_g$.

The bicategory of pointed 1-types is the total bicategory of this displayed bicategory.

▶ **Example 4.4** (`disp_fullsubbicat`). We can select the 0-cells of a bicategory B by attaching a property $P : \mathsf{B} \to \mathsf{hProp}$. Define a displayed bicategory D over B such that $\mathsf{D}_x :\equiv P(x)$, and the types of displayed 1-cells and 2-cells are the unit type. Now we define the full subbicategory of B with cells satisfying P to be the total bicategory of D.

We end this section with several general constructions of displayed bicategories.

▶ **Definition 4.5** (Various constructions of displayed bicategories).

1. (`disp_dirprod_bicat`) Given displayed bicategories D_1 and D_2 over a bicategory B, we construct the product $\mathsf{D}_1 \times \mathsf{D}_2$ over B. The 0-cells, 1-cells, and 2-cells are pairs of 0-cells, 1-cells, and 2-cells respectively.

2. (`sigma_bicat`) Given a displayed bicategory D over a base B and a displayed bicategory E over $\int\mathsf{D}$, we construct a displayed bicategory $\sum_{\mathsf{D}}\mathsf{E}$ over B as follows. The objects over $a:\mathsf{B}$ are pairs $(\bar{a},e)$, where $\bar{a}:\mathsf{D}_a$ and $e:\mathsf{E}_{(a,\bar{a})}$, the morphisms over $f:a\to b$ from $(\bar{a},e)$ to $(\bar{b},e')$ are pairs $(\bar{f},\varphi)$, where $\bar{f}:\bar{a}\xrightarrow{f}\bar{b}$ and $\varphi:e\xrightarrow{(f,\bar{f})}e'$, and similarly for 2-cells.
3. (`trivial_displayed_bicat`) Every bicategory D is, in a trivial way, a displayed bicategory over any other bicategory B. Its total bicategory is the direct product $\mathsf{B}\times\mathsf{D}$.
4. (`disp_cell_unit_bicat`) We say a displayed bicategory D over B is **chaotic** if, for each $\alpha:f\Rightarrow g$ and $\bar{f}:\bar{a}\xrightarrow{f}\bar{b}$ and $\bar{g}:\bar{a}\xrightarrow{g}\bar{b}$, the type $\bar{f}\overset{\alpha}{\Rightarrow}\bar{g}$ is contractible. Let B be a bicategory and suppose we have
   - a type D_0 and a type family D_1 on B as in Definition 4.1;
   - displayed 1-identities id_1 and compositions $(\cdot)$ of displayed 1-cells as Definition 4.1.

   Then we have an associated **chaotic displayed bicategory** $\hat{\mathsf{D}}(\mathsf{D}_0,\mathsf{D}_1,\mathrm{id}_1,(\cdot))$ over B by stipulating that the types of 2-cells are the unit type.

5 Displayed univalence

Given a bicategory B and a displayed bicategory D on B, our goal is to prove the univalence of $\int\mathsf{D}$ from conditions on B and D. For that, we develop the notion of *univalent displayed bicategories*. We start by defining displayed versions of invertible 2-cells.

▶ **Definition 5.1** (`is_disp_invertible_2cell`)**.** Given are a bicategory B and a displayed bicategory D over B. Suppose we have objects $a,b:\mathsf{B}_0$, two 1-cells $f,g:\mathsf{B}_1(a,b)$, and an invertible 2-cell $\theta:\mathsf{B}_2(f,g)$. Suppose that we also have $\bar{a}:\mathsf{D}_0(a)$, $\bar{b}:\mathsf{D}_0(b)$, $\bar{f}:\bar{a}\xrightarrow{f}\bar{b}$, $\bar{g}:\bar{a}\xrightarrow{g}\bar{b}$, and $\bar{\theta}:\bar{f}\overset{\theta}{\Rightarrow}\bar{g}$. Then we say $\bar{\theta}$ is **invertible** if we have $\bar{\gamma}:\bar{g}\overset{\theta^{-1}}{\Longrightarrow}\bar{f}$ such that $\bar{\theta}\bullet\bar{\gamma}$ and $\bar{\gamma}\bullet\bar{\theta}$ are identities modulo transport over the corresponding identity laws of θ.

A **displayed invertible 2-cell over** θ, where θ is an invertible 2-cell, is a pair of a displayed 2-cell $\bar{\theta}$ over θ and a proof that $\bar{\theta}$ is invertible. The type of displayed invertible 2-cells from $\bar{f}$ to $\bar{g}$ over θ is denoted by $\bar{f}\cong_\theta\bar{g}$.

Being a displayed invertible 2-cell is a proposition and the displayed 2-cell $\mathrm{id}_2(\bar{f})$ over $\mathrm{id}_2(f)$ is invertible. Next we define displayed adjoint equivalences.

▶ **Definition 5.2** (`disp_left_adjoint_equivalence`)**.** Given are a bicategory B and a displayed bicategory D over B. Suppose we have objects $a,b:\mathsf{B}_0$ and a 1-cell $f:\mathsf{B}_1(a,b)$ together with an adjoint equivalence structure A on f. We write r, η, ε for the right adjoint, unit, and counit of f respectively. Furthermore, suppose that we have $\bar{a}:\mathsf{D}_0(a)$,$\bar{b}:\mathsf{D}_0(b)$, and $\bar{f}:\bar{a}\xrightarrow{f}\bar{b}$. A **displayed adjoint equivalence structure** on $\bar{f}$ consists of
- A displayed 1-cell $\bar{r}:\bar{b}\xrightarrow{r}\bar{a}$;
- An invertible displayed 2-cell $\mathrm{id}_1(\bar{a})\overset{\eta}{\Rightarrow}\bar{f}\cdot\bar{r}$;
- An invertible displayed 2-cell $\bar{r}\cdot\bar{f}\overset{\varepsilon}{\Rightarrow}\mathrm{id}_1(\bar{b})$.

In addition, two laws reminiscent of those in Definition 2.4 need to be satisfied.

A **displayed adjoint equivalence** over the adjoint equivalence A is a pair of a displayed 1-cell $\bar{f}$ over f together with a displayed adjoint equivalence structure on $\bar{f}$. The type of displayed adjoint equivalences from $\bar{a}$ to $\bar{b}$ over f is denoted by $\bar{a}\simeq_f\bar{b}$.

The displayed 1-cell $\mathrm{id}_1(\bar{a})$ is a displayed adjoint equivalence over $\mathrm{id}_1(a)$.

Using these definitions, we define univalence of displayed bicategories similarly to univalence for ordinary bicategories. Again we separate it in a local and global condition.

▶ **Definition 5.3** (`DispUnivalence.v`). Let D be a displayed bicategory over B.

1. Let $a, b : \mathsf{B}$, and $\bar{a} : \mathsf{D}_a, \bar{b} : \mathsf{D}_b$. Let $f, g : a \to b$, let $p : f = g$, and let $\bar{f}$ and $\bar{g}$ be displayed morphisms over f and g respectively. Then we define a function

$$\mathsf{disp_idtoiso}^{2,1}_{p,\bar{f},\bar{g}} : \bar{f} =_p \bar{g} \to \bar{f} \cong_{\mathsf{idtoiso}^{2,1}_{f,g}(p)} \bar{g}$$

sending refl to the identity displayed isomorphism. We say that D is **locally univalent** if the map $\mathsf{disp_idtoiso}^{2,1}_{p,\bar{f},\bar{g}}$ is an equivalence for each p, $\bar{f}$, and $\bar{g}$.

2. Let $a, b : \mathsf{B}$, and $\bar{a} : \mathsf{D}_a, \bar{b} : \mathsf{D}_b$. Given $p : a = b$, we define a function

$$\mathsf{disp_idtoiso}^{2,0}_{p,\bar{a},\bar{b}} : \bar{a} =_p \bar{b} \to \bar{a} \simeq_{\mathsf{idtoiso}^{2,0}_{a,b}(p)} \bar{b}$$

sending refl to the identity displayed adjoint equivalence. We say that D is **globally univalent** if the map $\mathsf{disp_idtoiso}^{2,0}_{p,\bar{a},\bar{b}}$ is an equivalence for each p, $\bar{a}$, and $\bar{b}$.

3. (`disp_univalent_2`) We call D **univalent** if it is both locally and globally univalent.

Now we give the main theorem of this paper. It says that the total bicategory $\int_{\mathsf{B}} \mathsf{D}$ is univalent if B and D are.

▶ **Theorem 5.4** (`total_is_univalent_2`). *Let B be a bicategory and let D be a displayed bicategory over B. Then*

1. *$\int \mathsf{D}$ is locally univalent if B is locally univalent and D is locally univalent;*
2. *$\int \mathsf{D}$ is globally univalent if B is globally univalent and D is globally univalent.*

Proof. The main idea behind the proof is to characterize invertible 2-cells in the total bicategory as pairs of an invertible 2-cell p in the base bicategory, and a displayed invertible 2-cell over p. Concretely, for the local univalence, we factor $\mathsf{idtoiso}^{2,1}$ as a composition of the following equivalences:

$$\begin{array}{ccc} (f,\bar{f}) = (g,\bar{g}) & \xrightarrow{\mathsf{idtoiso}^{2,1}} & \mathsf{inv2cell}\big((f,\bar{f}),(g,\bar{g})\big) \\ {\scriptstyle w_1}\downarrow{\scriptstyle \sim} & & {\scriptstyle \sim}\uparrow{\scriptstyle w_3} \\ \sum_{(p:f=g)} \bar{f} =_p \bar{g} & \xrightarrow[w_2]{\sim} & \sum_{(p:\mathsf{inv2cell}(f,g))} \bar{f} \cong_p \bar{g} \end{array}$$

The map w_1 is just a characterization of paths in a sigma type. The map w_2 turns equalities into (displayed) invertible 2-cells, and it is an equivalence by local univalence of B and displayed local univalence of D. Finally, the map w_3 characterizes invertible 2-cells in the total bicategory.

The proof is similar in the case of global univalence. The most important step is the characterization of adjoint equivalences in the total bicategory.

$$(a,\bar{a}) \simeq (b,\bar{b}) \xrightarrow{\sim} \sum_{(p:a\simeq b)} \bar{a} \simeq_p \bar{b}. \qquad ◀$$

To check displayed univalence, it suffices to prove the condition in the case where p is reflexivity. This step, done by path induction, simplifies some proofs of displayed univalence.

▶ **Proposition 5.5.** *Given a displayed bicategory D over B, then D is univalent if the following maps are equivalences:*

- *(fiberwise_local_univalent_is_univalent_2_1)*

$$\mathsf{disp_idtoiso}^{2,1}_{\mathsf{refl}(f),\bar{f},\overline{f'}} : \bar{f} = \overline{f'} \to \bar{f} \cong_{\mathrm{id}_2(f)} \overline{f'}$$

- *(fiberwise_univalent_2_0_to_disp_univalent_2_0)*

$$\mathsf{disp_idtoiso}^{2,0}_{\mathsf{refl}(a),\bar{a},\overline{a'}} : \bar{a} = \overline{a'} \to \bar{a} \simeq_{\mathrm{id}_1(a)} \overline{a'}$$

Now we establish the univalence of several examples.

▶ **Example 5.6.** The following bicategories and displayed bicategories are univalent:

1. The category of pointed 1-types (see Example 4.3) is univalent (`p1types_univalent_2`).
2. The full subbicategory (see Example 4.4) of a univalent bicategory is univalent (`is_univalent_2_fullsubbicat`).
3. The product of univalent displayed bicategories (Definition 4.5, Item 1) is univalent (`is_univalent_2_dirprod_bicat`).
4. Given univalent displayed bicategories D_1 and D_2 on B and $\int \mathsf{D}_1$ respectively, the displayed bicategory $\sum_{\mathsf{D}_1} \mathsf{D}_2$ (Definition 4.5, Item 2) is univalent (`sigma_is_univalent_2`).

Lastly, we give a condition for when the chaotic displayed bicategory is univalent.

▶ **Proposition 5.7** (`disp_cell_unit_bicat_univalent_2`)**.** *Let* B *be a univalent bicategory. Given* $\mathsf{D} = (\mathsf{D}_0, \mathsf{D}_1, \mathrm{id}_1, (\cdot))$ *as in Definition 4.5, Item 4, such that* D_0 *is a set and* D_1 *is a family of propositions. Then the chaotic displayed bicategory on D is univalent if we have a map in the opposite direction of* $\mathsf{disp_idtoiso}^{2,0}$.

6 Univalence of Complicated Bicategories

In this section, we demonstrate the power of displayed bicategories on a number of complicated examples. We show the univalence of the bicategory of pseudofunctors between univalent bicategories and of univalent categories with families. In addition, we give two constructions to define univalent bicategories of algebras.

6.1 Pseudofunctors

As promised, we use displayed bicategories to prove Theorem 3.6. For the remainder, fix bicategories B and C such that C is univalent. Recall that a pseudofunctor consists of an action on 0-cells, 1-cells, 2-cells, a family of 2-cells witnessing the preservation of composition and identity 1-cells, such that a number of laws are satisfied. We call the 2-cells witnessing the preservation of composition and identity the *compositor* and *identitor* respectively.

To construct $\mathsf{Pseudo}(\mathsf{B}, \mathsf{C})$, we add structure to a base bicategory in several layers. This base bicategory consists of functions from B_0 to C_0. Each layer is given by a displayed bicategory on the total bicategory of the preceding layer. We start by defining a displayed bicategory of actions on 1-cells. On its total bicategory, we define three displayed bicategories: one for the preservation of composition, one for the preservation of identities, and one for the action on 2-cells. We take the product of these three and we finish by taking a full subbicategory with the required laws. To show the resulting bicategory is univalent, we show the base and each layer is univalent.

Now let us look at the formal definitions.

▶ **Definition 6.1** (`ps_base`)**.** The bicategory $\mathsf{Base}(\mathsf{B}, \mathsf{C})$ is defined as follows.

- The objects are maps $\mathsf{B}_0 \to \mathsf{C}_0$;
- The 1-cells from F_0 to G_0 are maps $\eta_0, \beta_0 : \prod_{(x:\mathsf{B}_0)} F_0(x) \to G_0(x)$;
- The 2-cells from η_0 to β_0 are maps $\Gamma : \prod_{(x:\mathsf{B}_0)} \eta_0(x) \Rightarrow \beta_0(x)$.

The operations are defined pointwise.

Next we define a displayed bicategory on $\mathsf{Base}(\mathsf{B},\mathsf{C})$. The displayed 0-cells are actions of pseudofunctors on 1-cells. The displayed 1-cells over η_0 are 2-cells witnessing the naturality of η_0. The displayed 2-cells over Γ are equalities which show that Γ is a modification.

▶ **Definition 6.2** (`map1cells_disp_bicat`)**.** We define a displayed bicategory $\mathsf{Map1D}(\mathsf{B},\mathsf{C})$ on $\mathsf{Base}(\mathsf{B},\mathsf{C})$ such that

- the objects over $F_0 : \mathsf{B}_0 \to \mathsf{C}_0$ are maps

$$F_1 : \prod_{(X,Y:\mathsf{B}_0)} \mathsf{B}_1(X,Y) \to \mathsf{C}_1(F_0(X), F_0(Y));$$

- the 1-cells over $\eta_0 : F_0(x) \to G_0(x)$ from F_1 to G_1 are invertible 2-cells

$$\eta_1 : \prod_{(X,Y:\mathsf{B}_0)(f:X\to Y)} \eta_0(X) \cdot G_1(f) \Rightarrow F_1(f) \cdot \eta_0(Y);$$

- the 2-cells over $\Gamma : \eta_0(x) \Rightarrow \beta_0(x)$ from η_1 to β_1 are equalities

$$\prod_{(X,Y:\mathsf{B}_0)(f:X\to Y)} \eta_1(f) \bullet (F_1(f) \triangleleft \Gamma(Y)) = (\Gamma(X) \triangleright G_1(f)) \bullet \beta_1(f).$$

We denote the total bicategory of $\mathsf{Map1D}(\mathsf{B},\mathsf{C})$ by $\mathsf{Map1}(\mathsf{B},\mathsf{C})$. Now we define three displayed bicategories over $\mathsf{Map1}(\mathsf{B},\mathsf{C})$. Each of them is defined as a chaotic displayed bicategory (Item 4 in Definition 4.5).

▶ **Definition 6.3** (`identitor_disp_cat`)**.** We define a displayed bicategory $\mathsf{MapId}(\mathsf{B},\mathsf{C})$ over $\mathsf{Map1}(\mathsf{B},\mathsf{C})$ as follows:

- The objects over (F_0, F_1) are identitors

$$F_i : \prod_{(X:\mathsf{B}_0)} \mathrm{id}_1(F_0(X)) \Rightarrow F_1(\mathrm{id}_1(X));$$

- The morphisms over (η_0, η_1) from F_i to G_i are equalities

$$\rho(\eta_0(X)) \bullet \lambda(\eta_0(X))^{-1} \bullet (F_i(X) \triangleright \eta_0(X)) = (\eta_0(X) \triangleleft G_i(X)) \bullet \eta_1(\mathrm{id}_1(X)).$$

▶ **Definition 6.4** (`compositor_disp_cat`)**.** We define a displayed bicategory $\mathsf{MapC}(\mathsf{B},\mathsf{C})$ over $\mathsf{Map1}(\mathsf{B},\mathsf{C})$ as follows:

- The objects over (F_0, F_1) are compositors

$$F_c : \prod_{(X,Y,Z:\mathsf{B}_0)(f:\mathsf{B}_1(X,Y))(g:\mathsf{B}_1(Y,Z))} F_1(f) \cdot F_1(g) \Rightarrow F_1(f \cdot g);$$

- The morphisms over (η_0, η_1) from F_c to G_c consists of equalities

$$\alpha \bullet (\eta_1(f) \triangleright G_1(g)) \bullet \alpha^{-1} \bullet (F_1(f) \triangleleft \eta_1(g)) \bullet \alpha \bullet (F_c \triangleright \eta_0(Z)) = (\eta_0(X) \triangleleft G_c) \bullet \eta_1(f \cdot g)$$

  for all $X, Y, Z : \mathsf{B}_0$, $f : \mathsf{B}_1(X,Y)$ and $g : \mathsf{B}_1(Y,Z)$.

▶ **Definition 6.5** (`map2cells_disp_cat`)**.** We define a displayed bicategory $\mathsf{Map2D}(\mathsf{B},\mathsf{C})$ over $\mathsf{Map1}(\mathsf{B},\mathsf{C})$ as follows:

- The objects over (F_0, F_1) are

$$F_2 : \prod_{(a,b:\mathsf{B}_0)(f,g:a\to b)} \mathsf{B}_2(f,g) \to F_1(f) \Rightarrow F_1(g);$$

- The morphisms over (η_0, η_1) from F_2 to G_2 consist of equalities

$$\prod_{(\theta : f \Rightarrow g)} (\eta_0(X) \lhd G_2(\theta)) \bullet \eta_1(g) = \eta_1(f) \bullet (F_2(\theta) \rhd \eta_0(Y)).$$

We denote the total category of the product of $\mathsf{Map2D}(\mathsf{B}, \mathsf{C})$, $\mathsf{MapId}(\mathsf{B}, \mathsf{C})$, and $\mathsf{MapC}(\mathsf{B}, \mathsf{C})$ by $\mathsf{RawPseudo}(\mathsf{B}, \mathsf{C})$. Note that its objects are of the form $((F_0, F_1), (F_2, F_i, F_c))$, its 1-cells are pseudotransformations, and its 2-cells are modifications. However, its objects are not yet pseudofunctors, because they also need to satisfy several laws.

▶ **Definition 6.6** (psfunctor_bicat)**.** We define the bicategory $\mathsf{Pseudo}(\mathsf{B}, \mathsf{C})$ as the full subbicategory of $\mathsf{RawPseudo}(\mathsf{B}, \mathsf{C})$ where the objects satisfy the following laws

- $F_2(\mathrm{id}_2(f)) = \mathrm{id}_2(F_1(f))$ and $F_2(f \bullet g) = F_2(f) \bullet F_2(g)$;
- $\lambda(F_1(f)) = (F_i(a) \rhd F_1(f)) \bullet F_c(\mathrm{id}_1(a), f) \bullet F_2(\lambda(f))$;
- $\rho(F_1(f)) = (F_1(f) \lhd F_i(b)) \bullet F_c(f, \mathrm{id}_1(b)) \bullet F_2(\rho(f))$;
- $(F_1(f) \bullet F_c(g, h)) \bullet F_c(f, g \cdot h) \bullet F_2(\alpha) = \alpha \bullet (F_c(f, g) \rhd F_1(h)) \bullet F_c(f \cdot g, h)$;
- $F_c(f, g_1) \bullet F_2(f \lhd \theta) = (F_1(f) \lhd F_2(\theta)) \bullet F_c(f, g_2)$;
- $F_c(f_1, g) \bullet F_2(\theta \rhd g) = (F_2(\theta) \rhd F_1(g)) \bullet F_c(f_2, g)$;
- $F_i(X)$ and $F_c(f, g)$ are invertible 2-cells.

Each displayed layer in this construction is univalent. In addition, if C is univalent, then so is $\mathsf{Base}(\mathsf{B}, \mathsf{C})$. Hence, Theorem 3.6 follows from repeated application of Theorem 5.4.

6.2 Algebraic Examples

Next, we consider two constructions to build bicategories of algebras. To illustrate their usage, we show how to define the bicategory of monads internal to a bicategory. Note that each monad has a 0-cell X and a 1-cell $X \to X$. This structure is encapsulated by *algebras of a pseudofunctor* [6].

▶ **Definition 6.7** (disp_alg_bicat)**.** Let B be a bicategory and let $F : \mathsf{Pseudo}(\mathsf{B}, \mathsf{B})$ be a pseudofunctor. We define a displayed bicategory $\mathsf{Alg}_{\mathsf{D}}(F)$.

- The objects over $a : \mathsf{B}$ are 1-cells $F(a) \to a$.
- The 1-cells over $f : \mathsf{B}_1(a, b)$ from $h_a : F(a) \to a$ to $h_b : F(b) \to b$ are invertible 2-cells $h_a \cdot f \Rightarrow F_1(f) \cdot h_b$.
- Given $f, g : \mathsf{B}_1(a, b)$, algebras $h_a : F(a) \to a$ and $h_b : F(b) \to b$, and h_f and h_g over f and g respectively, a 2-cell over $\theta : f \Rightarrow g$ is an equality

$$(h_a \lhd \theta) \bullet h_g = h_f \bullet (F_2(\theta) \rhd h_b).$$

We write $\mathsf{Alg}(F)$ for the total category of $\mathsf{Alg}_{\mathsf{D}}(F)$.

▶ **Theorem 6.8** (bicat_algebra_is_univalent_2)**.** *Let* B *be a bicategory and let* $F : \mathsf{Pseudo}(\mathsf{B}, \mathsf{B})$ *be a pseudofunctor. If* B *is univalent, then so is* $\mathsf{Alg}(F)$*.*

▶ **Example 6.9** (Example 4.3 cont'd)**.** The bicategory of pointed 1-types is the bicategory of algebras for the constant pseudofunctor $F(a) = 1$.

Define M_1 to be $\mathsf{Alg}(\mathrm{id}_0(\mathsf{B}))$. Objects of M_1 consist of an $X : \mathsf{B}_0$ and a 1-cell $X \to X$. These are not monads yet, because those are supposed to also have two 2-cells: the unit and multiplication. To add this structure, we define two displayed bicategories on M_1. Both are defined via a more general construction, for which we use that there is an identity pseudofunctor $\mathrm{id}_0(\mathsf{B}) : \mathsf{Pseudo}(\mathsf{B}, \mathsf{B})$ and that for all $F_1 : \mathsf{Pseudo}(\mathsf{B}_1, \mathsf{B}_2)$ and $F_2 : \mathsf{Pseudo}(\mathsf{B}_2, \mathsf{B}_3)$, we have a composition $F_1 \cdot F_2 : \mathsf{Pseudo}(\mathsf{B}_1, \mathsf{B}_3)$.

Before giving this construction, let us describe the setting. Suppose that we have a displayed bicategory D over some B. Our goal is to define a displayed bicategory over $\int D$ where the displayed 0-cells are certain 2-cells in B. We define the endpoints of these as natural 1-cells, so we use pseudotransfomations. The source of these is $\pi_D \cdot S$ for some $S : \mathsf{Pseudo}(\mathsf{B}, \mathsf{B})$, and the target is defined to be $\pi_D \cdot \mathrm{id}_0(\mathsf{B})$ where π_D is the projection from $\int D$ to B. Note that instead of π_D, we use $\pi_D \cdot \mathrm{id}_0(\mathsf{B})$, which is symmetric to the source $\pi_D \cdot S$. This allows us to construct such transformations by composing them. In addition, pseudotransformations $l, r : \pi_D \cdot S \to \pi_D \cdot \mathrm{id}_0(\mathsf{B})$ give, for each $(a, h_a) : \int D$, a 1-cell $l(a) : \mathsf{B}_1(S(a), a)$. The construction adds 2-cells from $l(a)$ to $r(a)$ and formally, we define the following displayed bicategory.

▶ **Definition 6.10** (`add_cell_disp_cat`)**.** Suppose that D is a displayed bicategory over B and given are $S : \mathsf{Pseudo}(\mathsf{B}, \mathsf{B})$ and $l, r : \pi_D \cdot S \to \pi_D \cdot \mathrm{id}_0(\mathsf{B})$. We use Item 4 in Definition 4.5 to define a displayed bicategory $\mathsf{Add2Cell}(\mathsf{D}, l, r)$ over $\int D$.

- Its objects over a are 2-cells $l(a) \Rightarrow r(a)$.
- The morphisms over $f : \mathsf{B}_1(a, b)$ from η_1 to η_2 are equalities

$$(\eta_1 \rhd \pi_D(f)) \bullet r(f) = l(f) \bullet (S_1(\pi_D(f)) \lhd \eta_2).$$

▶ **Theorem 6.11.** *The displayed bicategory* $\mathsf{Add2Cell}(\mathsf{D}, l, r)$ *is locally univalent (*`add_cell_disp_cat_univalent_2_1`*). Moreover, if* C *is locally univalent and* D *is locally univalent, then* $\mathsf{Add2Cell}(\mathsf{D}, l, r)$ *is globally univalent (*`add_cell_disp_cat_univalent_2_0`*).*

Let us show how to add the unit and multiplication to the structure. For that, we first need the following pseudotransformation.

▶ **Definition 6.12** (`alg_map`)**.** Let B be a bicategory, and let $F : \mathsf{Pseudo}(\mathsf{B}, \mathsf{B})$ be pseudofunctors. We define a pseudotransformation $\mathsf{h} : \pi_{\mathsf{Alg_D}F} \cdot F \to \pi_{\mathsf{Alg_D}F} \cdot \mathrm{id}_0(\mathsf{B})$. On objects (a, h_a), we define $\mathsf{h}(a, h_a) = h_a$ and on 1-cells (f, h_f), we define $\mathsf{h}(f, h_f) = h_f$.

To add the unit to the structure, we use Definition 6.10. For S, we take id_0, for l we take the identity transformation on $\pi_{\mathrm{id}_0} \cdot \mathrm{id}_0$ and for r we take h. The multiplication is done similarly, but instead we take $\mathsf{h} \cdot \mathsf{h}$ for l.

Let M_2 be the total bicategory of the product of these two displayed bicategories. To obtain actual monads, the structure needs to satisfy three laws, namely

- $\lambda(f)^{-1} \bullet (\eta \rhd f) \bullet \mu = \mathrm{id}_2(f)$;
- $\rho(f)^{-1} \bullet (f \lhd \eta) \bullet \mu = \mathrm{id}_2(f)$;
- $(f \lhd \mu) \bullet \mu = \alpha(f, f, f) \bullet (\mu \rhd f) \bullet \mu$.

We define $\mathsf{M}(\mathsf{B})$ to be the full subbicategory of M_2 with respect to these laws. From Theorems 6.8 and 6.11 and Example 5.6 we conclude:

▶ **Theorem 6.13** (`monad_is_univalent_2`)**.** *If* B *is univalent, then so is* $\mathsf{M}(\mathsf{B})$*.*

6.3 Categories with Families

Finally, we discuss the last example: the bicategory of (univalent) categories with families (CwFs) [9]. We follow the formulation by Fiore [12] and Awodey [4], which is already formalized in `UniMath` [3]: a CwF consists of a category C, two presheaves Ty and Tm on C, a morphism $p : \mathsf{Tm} \to \mathsf{Ty}$, and a representation structure for p.

However, rather than defining CwFs in one step, we use a stratified construction yielding the sought bicategory as the total bicategory of iterated displayed layers. The base bicategory is Cat (cf. Example 2.7). The second layer of data consists of two presheaves, each described by the following construction.

▶ **Definition 6.14** (disp_presheaf_bicat). Define the displayed bicategory PShD on Cat:

- The objects over C are functors from C^{op} to the univalent category Set;
- The 1-cells from $T : C \to \mathsf{Set}$ to $T' : D \to \mathsf{Set}$ over $F : C \to D$ are natural transformations from T to $F^{\mathsf{op}} \cdot T'$;
- The 2-cells from $\beta : T \Rightarrow F^{\mathsf{op}} \cdot T'$ to $\beta' : T \Rightarrow G^{\mathsf{op}} \cdot T'$ over $\gamma : F \Rightarrow G$ are equalities

$$\beta = \beta' \bullet (\gamma^{\mathsf{op}} \triangleright T').$$

Denote by CwF_1 the total category of the product of PShD with itself. An object in CwF_1 consists of a category C and two presheaves $\mathsf{Ty}, \mathsf{Tm} : C^{\mathsf{op}} \to \mathsf{Set}$. The third piece of data is a natural transformation between them.

▶ **Definition 6.15** (morphisms_of_presheaves_display). We define a displayed bicategory dCwF_2 on CwF_1 as the chaotic displayed bicategory (Item 4 in Definition 4.5) such that

- The objects over $(C, (\mathsf{Ty}, \mathsf{Tm}))$ are natural transformations from Ty to Tm.
- Suppose we have two objects $(C, (\mathsf{Ty}, \mathsf{Tm}))$ and $(C', (\mathsf{Ty}', \mathsf{Tm}'))$, two natural transformations $\mathsf{p} : \mathsf{Tm} \Rightarrow \mathsf{Ty}$ and $\mathsf{p}' : \mathsf{Tm}' \Rightarrow \mathsf{Ty}'$, and suppose we have a 1-cell f from $(C, (\mathsf{Ty}, \mathsf{Tm}))$ to $(C', (\mathsf{Ty}', \mathsf{Tm}'))$. Note that f consists of a functor $F : C \to C'$ and two transformations $\beta : \mathsf{Ty} \Rightarrow F^{\mathsf{op}} \circ \mathsf{Ty}'$ and $\beta' : \mathsf{Tm} \Rightarrow F^{\mathsf{op}} \circ \mathsf{Tm}'$. Then a 1-cell over f is an equality

$$p \bullet \beta = \beta' \bullet (F^{\mathsf{op}} \triangleleft p').$$

With dCwF_2 and the sigma construction from Item 2 in Definition 4.5, we get a displayed bicategory over Cat and we denote its total bicategory by CwF_2. As the last piece of data, we add the representation structure for the morphism p of presheaves.

▶ **Definition 6.16** (cwf_representation). Given a category C together with functors $\mathsf{Ty}, \mathsf{Tm} : C^{\mathsf{op}} \to \mathsf{Set}$ and a natural transformation $p : \mathsf{Tm} \Rightarrow \mathsf{Ty}$, we say $\mathsf{isCwF}(C, \mathsf{Ty}, \mathsf{Tm}, p)$ if for each $\Gamma : C$ and $A : \mathsf{Ty}(\Gamma)$, we have a representation of the fiber of p over A.

A detailed definition can be found in [3, Definition 3.1]. Since C is univalent, the type $\mathsf{isCwF}(C, \mathsf{Ty}, \mathsf{Tm}, \mathsf{p})$ is a proposition, and thus we define CwF as a full subbicategory of CwF_2.

▶ **Proposition 6.17** ([3, Lemma 4.3] , isaprop_cwf_representation). $\mathsf{isCwF}(C, \mathsf{Ty}, \mathsf{Tm}, \mathsf{p})$ *is a proposition.*

▶ **Definition 6.18** (cwf). We define CwF as the full subbicategory of dCwF_2 with isCwF.

▶ **Theorem 6.19** (cwf_is_univalent_2). CwF *is univalent.*

7 Conclusions and Further Work

In the present work, we studied univalent bicategories. Showing that a bicategory is univalent can be challenging; to simplify this task, we introduced displayed bicategories, which provide a way to modularly reason about complicated bicategories. We then demonstrated the usefulness of displayed bicategories by showing, using the displayed technology, that several complicated bicategories are univalent.

For the practical mechanization of mathematics in a computer proof assistant, two issues may arise when building complicated bicategories as the total bicategory of iterated displayed bicategories. Firstly, the structures may not be parenthesized as desired. This problem can be avoided or at least alleviated through a suitable use of the sigma construction of displayed bicategories (Item 2 in Definition 4.5). Secondly, "meaningless" terms of unit type may occur

in the cells of this bicategory. We are not aware of a way of avoiding these occurrences while still using displayed bicategories. However, both issues can be addressed through the definition of a suitable "interface" to the structures, in form of "builder" and projection functions, which build, or project a component out of, an instance of the structure. The interface hides the implementation details of the structure, and thus provides a welcome separation of concerns between mathematical and foundational aspects.

We have only started, in the present work, the development of bicategory theory in univalent foundations and its formalization. Our main goals for the future are

A bicategorical Rezk completion: to construct the free univalent bicategory associated to a bicategory. It will fundamentally use Definition 6.6 and Theorem 3.6.

Equivalence Principle: to show that identity is biequivalence for univalent bicategories.

More displayed machinery: to define and study displayed notions of pseudofunctors, biequivalences, etc over the respective notions in the base. In particular, the extra displayed machinery will allow us to build not just (univalent) bicategories layerwise, but also maps and equivalences between them.

The envisioned displayed machinery can also be used to study the semantics of higher inductive types (HITs). Using Definitions 6.7 and 6.10, we can define bicategories of algebras on a signature; its initial object is the HIT specified by the signature. The Rezk completion $\eta : \mathsf{Grpd}_{\mathsf{U}} \to \mathsf{1\text{-}Type}_{\mathsf{U}}$ from groupoids to 1-types can then be used to construct a biadjunction – obtained as the total biadjunction of a suitable displayed biadjunction – between algebras of 1-types and algebras of groupoids. To construct higher inductive 1-types, we just need to show that the groupoid model has HITs, which was proved by Dybjer and Moenclaey [10].

Displayed notions naturally appear in Clairambault and Dybjer's [8] pair of biequivalences $\mathbf{FL} \rightleftarrows \mathbf{CwF}^{\mathrm{I_{ext}},\Sigma}_{\mathrm{dem}}$ and $\mathbf{LCC} \rightleftarrows \mathbf{CwF}^{\mathrm{I_{ext}},\Sigma,\Pi}_{\mathrm{dem}}$ relating categories with families equipped with structure modelling type and term formers to finite limit categories and locally cartesian closed categories, respectively. Here, the latter biequivalence is an "extension" of the former; this can be made formal by a displayed biequivalence relating the Π-structure with the locally cartesian closed structure.

More generally, we aim to use the displayed machinery when extending to the bicategorical setting the comparison of different categorical structures for type theories started in [3].

References

1 Benedikt Ahrens, Krzysztof Kapulkin, and Michael Shulman. Univalent categories and the Rezk completion. *Mathematical Structures in Computer Science*, 25:1010–1039, 2015. doi:10.1017/S0960129514000486.

2 Benedikt Ahrens and Peter LeFanu Lumsdaine. Displayed Categories. *Logical Methods in Computer Science*, 15(1), 2019. doi:10.23638/LMCS-15(1:20)2019.

3 Benedikt Ahrens, Peter LeFanu Lumsdaine, and Vladimir Voevodsky. Categorical structures for type theory in univalent foundations. *Logical Methods in Computer Science*, 14(3), September 2018. doi:10.23638/LMCS-14(3:18)2018.

4 Steve Awodey. Natural models of homotopy type theory. *Mathematical Structures in Computer Science*, 28(2):241–286, 2018. doi:10.1017/S0960129516000268.

5 Jean Bénabou. Introduction to bicategories. In *Reports of the Midwest Category Seminar*, pages 1–77, Berlin, Heidelberg, 1967. Springer Berlin Heidelberg. doi:10.1007/BFb0074299.

6 Robert Blackwell, Gregory M Kelly, and A John Power. Two-dimensional monad theory. *Journal of Pure and Applied Algebra*, 59(1):1–41, 1989. doi:10.1016/0022-4049(89)90160-6.

7 Paolo Capriotti and Nicolai Kraus. Univalent higher categories via complete Semi-Segal types. *PACMPL*, 2(POPL):44:1–44:29, 2018. doi:10.1145/3158132.

8 Pierre Clairambault and Peter Dybjer. The biequivalence of locally cartesian closed categories and Martin-Löf type theories. *Mathematical Structures in Computer Science*, 24(6), 2014. doi:10.1017/S0960129513000881.

9 Peter Dybjer. Internal Type Theory. In Stefano Berardi and Mario Coppo, editors, *Types for Proofs and Programs, International Workshop TYPES'95, Torino, Italy, June 5-8, 1995, Selected Papers*, volume 1158 of *Lecture Notes in Computer Science*, pages 120–134. Springer, 1995. doi:10.1007/3-540-61780-9_66.

10 Peter Dybjer and Hugo Moeneclaey. Finitary Higher Inductive Types in the Groupoid Model. *Electr. Notes Theor. Comput. Sci.*, 336:119–134, 2018. doi:10.1016/j.entcs.2018.03.019.

11 Eric Finster and Samuel Mimram. A type-theoretical definition of weak ω-categories. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005124.

12 Marcelo Fiore. Discrete Generalised Polynomial Functors, 2012. Slides from talk given at ICALP 2012, http://www.cl.cam.ac.uk/~mpf23/talks/ICALP2012.pdf.

13 Martin Hofmann and Thomas Streicher. The groupoid interpretation of type theory. In *Twenty-five years of constructive type theory (Venice, 1995)*, volume 36 of *Oxford Logic Guides*, pages 83–111. Oxford Univ. Press, New York, 1998.

14 Chris Kapulkin and Peter LeFanu Lumsdaine. The Simplicial Model of Univalent Foundations (after Voevodsky), 2012. arXiv:1211.2851.

15 Ambroise Lafont, Tom Hirschowitz, and Nicolas Tabareau. Types are weak omega-groupoids, in Coq. Talk at TYPES 2018, Link to online abstract (pdf).

16 Tom Leinster. Basic Bicategories, 1998. arXiv:math/9810017.

17 The Coq development team. *The Coq Proof Assistant*, 2018. Version 8.8.

18 nLab authors. Bicategory, December 2018. Revision 43.

19 Andrew M. Pitts. Categorical Logic. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science, Volume 5. Algebraic and Logical Structures*, chapter 2, pages 39–128. Oxford University Press, 2000.

20 Paul Taylor. *Practical Foundations of Mathematics*, volume 59 of *Cambridge studies in advanced mathematics*. Cambridge University Press, 1999.

21 The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. https://homotopytypetheory.org/book, Institute for Advanced Study, 2013.

22 Vladimir Voevodsky, Benedikt Ahrens, Daniel Grayson, et al. UniMath — a computer-checked library of univalent mathematics. Available at https://github.com/UniMath/UniMath.

Modular Specification of Monads Through Higher-Order Presentations

Benedikt Ahrens
University of Birmingham, United Kingdom
B.Ahrens@cs.bham.ac.uk

André Hirschowitz
Université Côte d'Azur, CNRS, LJAD, Nice, France
ah@unice.fr

Ambroise Lafont
IMT Atlantique, Inria, LS2N CNRS, Nantes, France
ambroise.lafont@inria.fr

Marco Maggesi
Università degli Studi di Firenze, Italy
marco.maggesi@unifi.it

Abstract

In their work on second-order equational logic, Fiore and Hur have studied presentations of simply typed languages by generating binding constructions and equations among them. To each pair consisting of a binding signature and a set of equations, they associate a category of "models", and they give a monadicity result which implies that this category has an initial object, which is the language presented by the pair.

In the present work, we propose, for the untyped setting, a variant of their approach where monads and modules over them are the central notions. More precisely, we study, for monads over sets, presentations by generating ("higher-order") operations and equations among them. We consider a notion of 2-signature which allows to specify a monad with a family of binding operations subject to a family of equations, as is the case for the paradigmatic example of the lambda calculus, specified by its two standard constructions (application and abstraction) subject to β- and η-equalities. Such a 2-signature is hence a pair (Σ,E) of a binding signature Σ and a family E of equations for Σ. This notion of 2-signature has been introduced earlier by Ahrens in a slightly different context.

We associate, to each 2-signature (Σ, E), a category of "models of (Σ, E)"; and we say that a 2-signature is "effective" if this category has an initial object; the monad underlying this (essentially unique) object is the "monad specified by the 2-signature". Not every 2-signature is effective; we identify a class of 2-signatures, which we call "algebraic", that are effective.

Importantly, our 2-signatures together with their models enjoy "modularity": when we glue (algebraic) 2-signatures together, their initial models are glued accordingly.

We provide a computer formalization for our main results.

2012 ACM Subject Classification Theory of computation → Algebraic language theory

Keywords and phrases free monads, presentation of monads, initial semantics, signatures, syntax, monadic substitution, computer-checked proofs

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.6

Supplement Material Computer-checked proofs with compilation instructions on `https://github.com/UniMath/largecatmodules/tree/50fd617`.

Funding This work has partly been funded by the CoqHoTT ERC Grant 637339. This material is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-17-1-0363.
Benedikt Ahrens: Ahrens acknowledges the support of the Centre for Advanced Study (CAS) in Oslo, Norway, which funded and hosted the research project *Homotopy Type Theory and Univalent Foundations* during the 2018/19 academic year.
Marco Maggesi: Supported by GNSAGA-INdAM and MIUR.

© Benedikt Ahrens, André Hirschowitz, Ambroise Lafont, and Marco Maggesi;
licensed under Creative Commons License CC-BY

Acknowledgements We thank Paige R. North for a valuable hint regarding preservation of epimorphisms. We also thank the referees for their careful reading and thoughtful and constructive criticism.

1 Introduction

The present work is devoted to the study of presentations of monads on the category of sets. More precisely, there is a well established theory of presentations of monads through generating (first-order) operations equipped with relations among the corresponding derived operations. Here we propose a counterpart of this theory, where we consider generation of monads by *binding operations*. Various algebraic structures generated by binding operations have been considered by many, going back at least to Fiore, Plotkin, and Turi [10], Gabbay and Pitts [12], and Hofmann [18]. Every such operation has a binding arity, which is a sequence of non-negative integers. For example, the binding arity of the application operation of the lambda calculus is $(0, 0)$: it takes two arguments without binding any variable in them, while the abstraction operation on the monad of the lambda calculus has binding arity (1), as it binds one variable in its single argument. For each family Σ of binding arities, there is a generated "free" monad $\hat{\Sigma}$ on Set which maps a set of free variables X to the set of terms $\hat{\Sigma}(X)$ taking variables in X.

If $p : \hat{\Sigma} \to R$ is a monad epimorphism, we understand that R is generated by a family of operations whose binding arities are given by Σ, subject to suitable identifications. In particular, for $\Sigma := ((0, 0), (1))$, $\hat{\Sigma}$ may be understood as the monad LC of syntactic terms of the lambda calculus, and we have an obvious epimorphism $p : \hat{\Sigma} \to \mathsf{LC}_{\beta\eta}$, where $\mathsf{LC}_{\beta\eta}$ is the monad of lambda-terms modulo β and η. In order to manage such equalities, the approach in the first-order case suggests to identify p as the coequalizer of a double arrow from T to $\hat{\Sigma}$ where T is again a "free" monad. Let us see what comes out when we attempt to find such an encoding for the β-equality of the monad $\mathsf{LC}_{\beta\eta}$. It should say that for each set X, the following two maps from $\hat{\Sigma}(X + \{*\}) \times \hat{\Sigma}(X)$ to $\hat{\Sigma}(X)$,

- $(t, u) \mapsto \mathsf{app}(\mathsf{abs}(t), u)$
- $(t, u) \mapsto t[* \mapsto u]$

are equal. Here a problem occurs, namely that the above collections of maps, which can be understood as a morphism of functors, cannot be understood as a morphism of monads. Notably, they do not send variables to variables.

On the other hand, we observe that the members of our equations, which are not morphisms of monads, commute with substitution, and hence are more than morphisms of functors: indeed they are morphisms of *modules over* $\hat{\Sigma}$. (In Section 2, we recall briefly what modules over a monad are.) Accordingly, a (second-order) presentation for a monad R could be a diagram

$$T \overset{f}{\rightrightarrows} \hat{\Sigma} \overset{p}{\longrightarrow} R \tag{1}$$

where Σ is a binding signature, $\hat{\Sigma}$ is the associated free monad, T is a module over $\hat{\Sigma}$, f is a pair of morphisms of modules over $\hat{\Sigma}$, and p is a monad epimorphism. And now we are faced with the task of finding a condition meaning something like "p is the coequalizer of f"[1]. To this end, we introduce the category Mon^{Σ} "of models of Σ", whose objects are monads "equipped with an action of Σ". Of course $\hat{\Sigma}$ is equipped with such an action which turns it into the initial object. Next, we define the full subcategory of models satisfying the equation

[1] This cannot be the case stricto sensu since f is a pair of module morphisms while p is a monad morphism.

f, and require R to be the initial object therein. Our definition is suited for the case where the equation f is parametric in the model: this means that now T and f are functions of the model S, and $f(S) = (u(S), v(S))$ is a pair of S-module morphisms from $T(S)$ to S. We say that S satisfies the equation f if $u(S) = v(S)$. Generalizing the case of one equation to the case of a family of equations yields the notion of 2-signature already introduced by Ahrens [1] in a slightly different context.

Now we are ready to formulate our main problem: given a 2-signature (Σ, E), where E is a family of parametric equations as above, does the subcategory of models of Σ satisfying the family of equations E admit an initial object?

We answer positively for a large subclass of 2-signatures which we call *algebraic* 2-signatures (see Theorem 32).

This provides a construction of a monad from an algebraic 2-signature, and we prove furthermore (see Theorem 27) that this construction is *modular*, in the sense that merging two extensions of 2-signatures corresponds to building an amalgamated sum of initial models. This is analogous to our previous result for 1-signatures shown in [2, Theorem 32].

As expected, our initiality property generates a recursion principle which is a recipe allowing us to specify a morphism from the presented monad to any given other monad.

We give various examples of monads arising “in nature” that can be specified via an algebraic 2-signature (see Section 6), and we also show through a simple example how our recursion principle applies (see Section 7).

Computer-checked formalization. This work is accompanied by a computer-checked formalization of the main results, based on the formalization of our previous work [2]. We work over the UniMath library [27], which is implemented in the proof assistant Coq [23]. The formalization consists of about 9,500 lines of code, and can be consulted on `https://github.com/UniMath/largecatmodules`. A guide is given in the README, and a summary of our formalization is available at `https://initialsemantics.github.io/doc/50fd617/Modules.SoftEquations.Summary.html`.

For the purpose of this article, we refer to a fixed version of our library, with the short hash 50fd617. This version compiles with version 10839ee of `UniMath`.

Throughout the article, statements are annotated with their corresponding identifiers in the formalization. These identifiers are also hyperlinks to the online documentation stored at `https://initialsemantics.github.io/doc/50fd617/index.html`.

Related work. The present work follows a previous work of ours [2] where we study a slightly different kind of presentation of monads. Specifically, in [2], we treat a class of 1-signatures which can be understood as quotients of algebraic 1-signatures. This should amount to considering a specific kind of equations, as suggested in Section 6.2, where we recover, in the current setting, all the examples given there.

Ahrens [1] introduces the notion of 2-signature which we consider here, in the slightly different context of (relative) monads on preordered sets, where the preorder models the reduction relation. In some sense, our result tackles the technical issue of quotienting the initial (relative) monad constructed in [1] by the preorder.

In a classical paper, Barr [3] explained the construction of the “free monad” generated by an endofunctor[2]. In another classical paper, Kelly and Power [19] explained how any finitary

[2] Fiore and Saville [11] give an enlightening generalization of the construction by Barr.

monad can be presented as a coequalizer of free monads[3]. There, free monads correspond to our initial models of an algebraic 1-signature without any binding construction.

As mentioned above, the present work is also closely related to that of Fiore and collaborators:

- Our notion of equations and that of model for them seem very close to the notion of equational systems and that of algebra for them in [7]: in particular, the preservation of epimorphisms, which occurs in their construction of inductive free algebras for equational systems, appears here in our definition of elementary equation. It would be interesting to understand formal connections between the two approaches.
- In [8], Fiore and Hur introduce a notion of equation based on syntax with *meta-variables*: essentially, a specific syntax, say, $T := T(M, X)$ considered there depends on two contexts: a meta-context M, and an object-context X. The terms of the actual syntax are then those terms $t \in T(\emptyset, X)$ in an empty meta-context. An equation for T is, simply speaking, a pair of terms in the same pair of contexts. Transferring an equation to any model of the underlying algebraic 1-signature is done by induction on the syntax with meta-variables. The authors show a monadicity theorem which straightforwardly implies an initiality result very similar to ours. That monadicity result is furthermore an instance of a more general theorem by Fiore and Mahmoud [9, Theorem 6.2].
- Translations between languages similar to the translation we present in Section 7 are also studied in [9]. Here again, it would be interesting to understand formal connections.
- At this stage, our work only concerns untyped syntax, but we anticipate it will generalize to the sorted setting as in [8] (see also the more general [6]).

Furthermore, Hamana [15] proposes initial algebra semantics for "binding term rewriting systems", based on Fiore, Plotkin, and Turi's presheaf semantics of variable binding and Lüth and Ghani's monadic semantics of term rewriting systems [21].

The alternative *nominal* approach to binding syntax initiated by Gabbay and Pitts [12] has been actively studied[4]. We highlight some contributions:

- Clouston [4] discusses signatures, structures (a.k.a. models), and equations over signatures in nominal style.
- Fernández and Gabbay [5] study signatures and equational theories as well as rewrite theories over signatures.
- Kurz and Petrisan [20] study closure properties of subcategories of algebras under quotients, subalgebras, and products. They characterize full subcategories closed under these operations as those that are definable by equations. They also show that the signature of the lambda calculus is effective, and study the subcategory of algebras of that signature specified by the β- and η-equations.

2 Categories of modules over monads

In this section, we recall the notions of monad and module over a monad, as well as some constructions of modules. We restrict our attention to the category Set of sets, although most definitions are straightforwardly generalizable. See [17] for a more extensive introduction.

A **monad** (over Set) is a triple $R = (R, \mu, \eta)$ given by a functor $R\colon \mathsf{Set} \longrightarrow \mathsf{Set}$, and two natural transformations $\mu\colon R \cdot R \longrightarrow R$ and $\eta\colon I \longrightarrow R$ such that the well-known monadic laws hold. A **monad morphism** to another such monad (R', μ', η') is a natural

[3] Their work has been applied to various more general contexts (e.g. [25]).
[4] The approaches by Fiore and collaborators and Gabbay and Pitts [12] are nicely compared by Power [24], who also comments on some generalization of the former approach.

transformation $f : R \to R'$ that commutes with the monadic structure. The category of monads is denoted by Mon.

Let R be a monad. A **(left) R-module**[5] is given by a functor $M\colon \mathsf{Set} \longrightarrow \mathsf{Set}$ equipped with a natural transformation $\rho\colon M \cdot R \longrightarrow M$, called *module substitution*, which is compatible with the monad composition and identity:

$$\rho \circ \rho R = \rho \circ M\mu, \qquad \rho \circ M\eta = 1_M.$$

Let $f\colon R \longrightarrow S$ be a morphism of monads and M an S-module. The module substitution $M \cdot R \stackrel{Mf}{\longrightarrow} M \cdot S \stackrel{\rho}{\longrightarrow} M$ turns M into an R-module f^*M, called **pullback of M along f**.

A natural transformation of R-modules $\varphi\colon M \longrightarrow N$ is **linear** if it is compatible with module substitution on either side, that is, if $\varphi \circ \rho^M = \rho^N \circ \varphi R$. Modules over R and their morphisms form a category denoted $\mathsf{Mod}(R)$, which is complete and cocomplete: limits and colimits are computed pointwise.

We define the **total module category** $\int_R \mathsf{Mod}(R)$ as follows: its objects are pairs (R, M) of a monad R and an R-module M. A morphism from (R, M) to (S, N) is a pair (f, m) where $f\colon R \longrightarrow S$ is a morphism of monads, and $m\colon M \longrightarrow f^*N$ is a morphism of R-modules. The category $\int_R \mathsf{Mod}(R)$ comes equipped with a forgetful functor to the category of monads, given by the projection $(R, M) \mapsto R$. This functor is a Grothendieck fibration with fiber $\mathsf{Mod}(R)$ over R. In particular, any monad morphism $f : R \longrightarrow S$ gives rise to a functor $f^*\colon \mathsf{Mod}(S) \longrightarrow \mathsf{Mod}(R)$ which preserves limits and colimits.

▶ **Example 1.** We give some important examples of modules:

1. Every monad R is a module over itself, which we call the **tautological module**.
2. For any functor $F\colon \mathsf{Set} \longrightarrow \mathsf{Set}$ and any R-module $M\colon \mathsf{Set} \longrightarrow \mathsf{Set}$, the composition $F \cdot M$ is an R-module (in the evident way).
3. For every set W we denote by $\underline{W}\colon \mathsf{Set} \longrightarrow \mathsf{Set}$ the constant functor $\underline{W} := X \mapsto W$. Then $\underline{W}$ is trivially an R-module since $\underline{W} = \underline{W} \cdot R$.
4. Given an R-module M, the R-module M' is defined on objects by $M'(X) := M(X + \{*\})$, and with the obvious module structure. Derivation yields an endofunctor on $\mathsf{Mod}(R)$ that is right adjoint to the functor $M \mapsto M \times R$, "product with the tautological module". Details are given, e.g., in [2, Section 2.3].
5. Derivation can be iterated. Given a list of non negative integers $(a) = (a_1, \ldots, a_n)$ and a left module M over a monad R, we denote by $M^{(a)} = M^{(a_1,\ldots,a_n)}$ the module $M^{(a_1)} \times \cdots \times M^{(a_n)}$, with $M^{()} = 1$ the final module.

3 1-signatures and their models

In this section, we review the notion of 1-signature studied in detail in [2] – there only called "signature".

A **1-signature** is a section of the forgetful functor from the category $\int_R \mathsf{Mod}(R)$ to the category Mon. A **morphism between two 1-signatures** $\Sigma_1, \Sigma_2\colon \mathsf{Mon} \longrightarrow \int_R \mathsf{Mod}(R)$ is a natural transformation $m\colon \Sigma_1 \longrightarrow \Sigma_2$ which, post-composed with the projection $\int_R \mathsf{Mod}(R) \longrightarrow \mathsf{Mon}$, is the identity. The category of 1-signatures is denoted by $\mathsf{1\text{-}Sig}$.

Limits and colimits of 1-signatures can be easily constructed pointwise: the category of 1-signatures is complete and cocomplete.

[5] The analogous notion of *right* R-module is not used in this work, we hence simply write "R-module" instead of "left R-module" for brevity.

Table 1 Examples of 1-signatures.

Hypotheses	On objects	Notation of the 1-signature
	$R \mapsto R$	Θ
Σ 1-signature, F functor	$R \mapsto F \cdot \Sigma(R)$	$F \cdot \Sigma$
	$R \mapsto 1_R$	1
Σ, Ψ 1-signatures	$R \mapsto \Sigma(R) \times \Psi(R)$	$\Sigma \times \Psi$
Σ, Ψ 1-signatures	$R \mapsto \Sigma(R) + \Psi(R)$	$\Sigma + \Psi$
	$R \mapsto R'$	Θ'
$n \in \mathbb{N}$	$R \mapsto R^{(n)}$	$\Theta^{(n)}$
$(a) = (a_1, \ldots, a_n) \in \mathbb{N}^n$	$R \mapsto R^{(a)} = R^{(a_1)} \times \ldots \times R^{(a_n)}$	$\Theta^{(a)}$ **elementary signatures**

Table 1 lists important examples of 1-signatures. An **algebraic 1-signature** is a (possibly infinite) coproduct of elementary signatures (defined in Table 1). For instance, the algebraic 1-signature of the lambda calculus is $\Sigma_{\mathsf{LC}} = \Theta^2 + \Theta'$.

Given a monad R over Set, we define an **action of the 1-signature Σ in R** to be a module morphism from $\Sigma(R)$ to R. For example, the application $\mathsf{app}\colon \mathsf{LC}^2 \longrightarrow \mathsf{LC}$ is an action of the elementary 1-signature Θ^2 into the monad LC of syntactic lambda calculus. The abstraction $\mathsf{abs}\colon \mathsf{LC}' \longrightarrow \mathsf{LC}$ is an action of the elementary 1-signature Θ' into the monad LC. Then $[\mathsf{app}, \mathsf{abs}] : \mathsf{LC}^2 + \mathsf{LC}' \longrightarrow \mathsf{LC}$ is an action of the algebraic 1-signature of the lambda-calculus $\Theta^2 + \Theta'$ into the monad LC.

Given a 1-signature Σ, we build the category Mon^Σ of **models of Σ** as follows. Its objects are pairs (R, r) of a monad R equipped with an action $r : \Sigma(R) \to R$ of Σ. A morphism from (R, r) to (S, s) is a morphism of monads $m : R \to S$ making the following diagram of R-modules commutes:

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

Let $f\colon \Sigma \longrightarrow \Psi$ be a morphism of 1-signatures and $\mathcal{R} = (R, r)$ a model of Ψ. The linear morphism $\Sigma(R) \xrightarrow{f(R)} \Psi(R) \xrightarrow{r} R$ defines an action of Σ in R. The induced model of Σ is called **pullback** of $\mathcal{R}$ along f and noted $f^*\mathcal{R}$.

The **total category** $\int_\Sigma \mathsf{Mon}^\Sigma$ of models is defined as follows:

- An object of $\int_\Sigma \mathsf{Mon}^\Sigma$ is a triple (Σ, R, r) where Σ is a 1-signature, R is a monad, and r is an action of Σ in R.
- A morphism in $\int_\Sigma \mathsf{Mon}^\Sigma$ from (Σ_1, R_1, r_1) to (Σ_2, R_2, r_2) consists of a pair (i, m) of a 1-signature morphism $i : \Sigma_1 \longrightarrow \Sigma_2$ and a morphism m of Σ_1-models from (R_1, r_1) to $(R_2, i^*(r_2))$.

The forgetful functor $\int_\Sigma \mathsf{Mon}^\Sigma \to \mathsf{Sig}$ is a Grothendieck fibration.

Given a 1-signature Σ, the initial object in Mon^Σ, if it exists, is denoted by $\hat{\Sigma}$. In this case, the 1-signature Σ is said **effective**[6].

▶ **Theorem 2** ([16, Theorems 1 and 2]). *Algebraic 1-signatures are effective.*

[6] In our previous work [2], we call **representable** any 1-signature Σ that has an initial model, called a **representation** of Σ, or **syntax generated by** Σ.

4 2-Signatures and their models

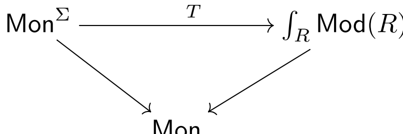
In this section we study *2-signatures* and *models of 2-signatures.* A 2-signature is a pair of a 1-signature and a family of *equations* over it.

4.1 Equations

Our equations are those of Ahrens [1]: they are parallel module morphisms parametrized by the models of the underlying 1-signature. The underlying notion of 1-model is essentially the same as in [1], even if, there, such equations are interpreted instead as *inequalities.*

Throughout this subsection, we fix a 1-signature Σ, that we instantiate in the examples.

▶ **Definition 3.** *We define a* $\boldsymbol{\Sigma}$***-module*** *to be a functor* T *from the category of models of* Σ *to the category* $\int_R \mathsf{Mod}(R)$ *commuting with the forgetful functors to the category* Mon *of monads,*



▶ **Example 4.** To each 1-signature Ψ is associated, by precomposition with the projection from Mon^Σ to Mon, a Σ-module still denoted Ψ. All the Σ-modules occurring in this work arise in this way from 1-signatures; in other words, they do not depend on the action of the 1-model. In particular, we have the **tautological Σ-module** Θ, and, more generally, for any natural number $n \in \mathbb{N}$, a Σ-module $\Theta^{(n)}$. Also we have another fundamental Σ-module (arising in this way from) Σ itself.

▶ **Definition 5.** *Let* S *and* T *be* Σ*-modules. We define a* ***morphism of*** $\boldsymbol{\Sigma}$***-modules*** *from* S *to* T *to be a natural transformation from* S *to* T *which becomes the identity when postcomposed with the forgetful functor* $\int_R \mathsf{Mod}(R) \to \mathsf{Mon}$*.*

▶ **Example 6.** Each 1-signature morphism $\Psi \to \Phi$ upgrades into a morphism of Σ-modules. Further in that vein, there is a morphism of Σ-modules $\tau^\Sigma : \Sigma \to \Theta$. It is given, on a model (R, m) of Σ, by $m : \Sigma(R) \to R$. (Note that it does not arise from a morphism of 1-signatures.) When the context is clear, we write simply τ for this morphism, and call it the **tautological morphism of Σ-modules**.

▶ **Proposition 7.** *Our* Σ*-modules and their morphisms, with the obvious composition and identity, form a category.*

▶ **Definition 8.** *We define a* Σ*-equation to be a pair of parallel morphisms of* Σ*-modules. We also write* $e_1 = e_2$ *for the* Σ*-equation* $e = (e_1, e_2)$*.*

▶ **Example 9** (Commutativity of a binary operation)**.** Here we instantiate our fixed 1-signature as follows: $\Sigma := \Theta \times \Theta$. In this case, we say that τ is the (tautological) binary operation. Now we can formulate the usual law of commutativity for this binary operation.

We consider the morphism of 1-signatures $\mathsf{swap} : \Theta^2 \longrightarrow \Theta^2$ that exchanges the two components of the direct product. Again by Example 6, we have an induced morphism of Σ-modules, still denoted swap.

Then, the Σ-equation for commutativity is given by the two morphisms of Σ-modules

$$\Theta^2 \xrightarrow{\mathsf{swap}} \Theta^2 \xrightarrow{\tau} \Theta$$
$$\Theta^2 \xrightarrow[\tau]{} \Theta$$

See also Section 6.1 where we explain in detail the case of monoids.

For the example of the lambda calculus with β- and η-equality (given in Example 11), we need to introduce *currying*:

▶ **Definition 10.** *By abstracting over the base monad R the adjunction in the category of R-modules of Example 1, item 4, we can perform* ***currying*** *of morphisms of 1-signatures: given a morphism of signatures $\Sigma_1 \times \Theta \to \Sigma_2$ it produces a new morphism $\Sigma_1 \to \Sigma_2'$. By Example 4, currying acts also on morphisms of Σ-modules.*

Conversely, given a morphism of 1-signatures (resp. Σ-modules) $\Sigma_1 \to \Sigma_2'$, we can define the ***uncurryied*** *map $\Sigma_1 \times \Theta \to \Sigma_2$.*

▶ **Example 11** (β- and η-conversions)**.** Here we instantiate our fixed 1-signature as follows: $\Sigma_{\mathsf{LC}} := \Theta \times \Theta + \Theta'$. This is the 1-signature of the lambda calculus. We break the tautological Σ-module morphism into its two pieces, namely $\mathsf{app} := \tau \circ \mathsf{inl} : \Theta \times \Theta \longrightarrow \Theta$ and $\mathsf{abs} := \tau \circ \mathsf{inr} : \Theta' \longrightarrow \Theta$. Applying currying to app yields the morphism $\mathsf{app}_1 : \Theta \longrightarrow \Theta'$ of Σ_{LC}-modules. The usual β and η relations are implemented in our formalism by two Σ_{LC}-equations that we call e_β and e_η respectively:

$$e_\beta : \begin{array}{l} \Theta' \xrightarrow{\mathsf{abs}} \Theta \xrightarrow{\mathsf{app}_1} \Theta' \\ \Theta' \xrightarrow[1]{} \Theta' \end{array} \quad \text{and} \quad e_\eta : \begin{array}{l} \Theta \xrightarrow{\mathsf{app}_1} \Theta' \xrightarrow{\mathsf{abs}} \Theta \\ \Theta \xrightarrow[1]{} \Theta \end{array}$$

4.2 2-signatures and their models

▶ **Definition 12.** *A* ***2-signature*** *is a pair (Σ, E) of a 1-signature Σ and a family E of Σ-equations.*

▶ **Example 13.** The 2-signature for a commutative binary operation is $(\Theta^2, \tau \circ \mathsf{swap} = \tau)$ (cf. Example 9).

▶ **Example 14.** The 2-signature of the lambda calculus modulo β- and η-equality is $\Upsilon_{\mathsf{LC}_{\beta\eta}} = (\Theta \times \Theta + \Theta', \{e_\beta, e_\eta\})$, where e_β, e_η are the Σ_{LC}-equations defined in Example 11.

▶ **Definition 15** (`satisfies_equation`)**.** *We say that a model M of Σ* ***satisfies the Σ-equation $e = (e_1, e_2)$*** *if $e_1(M) = e_2(M)$. If E is a family of Σ-equations, we say that a model M of Σ* ***satisfies E*** *if M satisfies each Σ-equation in E.*

▶ **Definition 16.** *Given a monad R and a 2-signature $\Upsilon = (\Sigma, E)$, an* ***action of Υ in*** *R is an action of Σ in R such that the induced 1-model satisfies all the equations in E.*

▶ **Definition 17** (`category_model_equations`)**.** *For a 2-signature (Σ, E), we define the* ***category*** *$\mathsf{Mon}^{(\Sigma,E)}$* ***of models of*** *$(\Sigma, E)$ to be the full subcategory of the category of models of Σ whose objects are models of Σ satisfying E, or equivalently, monads equipped with an action of (Σ, E).*

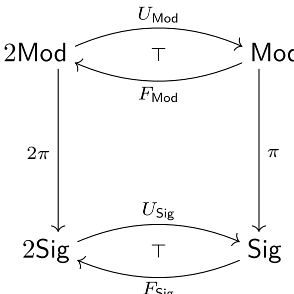
▶ **Example 18.** A model of the 2-signature $\Upsilon_{\mathsf{LC}_{\beta\eta}} = (\Theta \times \Theta + \Theta', \{e_\beta, e_\eta\})$ is given by a model $(R, \mathsf{app}^R : R \times R \to R, \mathsf{abs}^R : R' \to R)$ of the 1-signature Σ_{LC} such that $\mathsf{app}_1^R \cdot \mathsf{abs}^R = 1_{R'}$ and $\mathsf{abs}^R \cdot \mathsf{app}_1^R = 1_R$ (see Example 11).

▶ **Definition 19.** *A 2-signature* (Σ, E) *is said to be* ***effective*** *if its category of models* $\mathsf{Mon}^{(\Sigma,E)}$ *has an initial object, denoted* $\widehat{(\Sigma, E)}$.

In Section 4.4, we aim to find sufficient conditions for a 2-signature (Σ, E) to be effective.

4.3 Modularity for 2-signatures

In this section, we define the category $\mathsf{2Sig}$ of 2-signatures and the category $\mathsf{2Mod}$ of models of 2-signatures, together with functors that relate them with the categories of 1-signatures and 1-models. The situation is summarized in the commutative diagram of functors



where

- 2π is a Grothendieck fibration;
- π is the Grothendieck fibration defined in [2, Section 5.2];
- U_{Sig} is a coreflection and preserves colimits; and
- U_{Mod} is a coreflection.

As a simple consequence of this data, we obtain, in Theorem 27, a *modularity* result in the sense of Ghani, Uustalu, and Hamana [13]: it explains how the initial model of an amalgamated sum of 2-signatures is the amalgamation of the initial model of the summands[7].

We start by defining the category $\mathsf{2Sig}$ of 2-signatures:

▶ **Definition 20** (`TwoSig_category`). *Given 2-signatures* (Σ_1, E_1) *and* (Σ_2, E_2), *a* ***morphism of 2-signatures from*** $\boldsymbol{(\Sigma_1, E_1)}$ ***to*** $\boldsymbol{(\Sigma_2, E_2)}$ *is a morphism of 1-signatures* $m : \Sigma_1 \to \Sigma_2$ *such that for any model* M *of* Σ_2 *satisfying* E_2, *the* Σ_1*-model* m^*M *satisfies* E_1.

These morphisms, together with composition and identity inherited from 1-signatures, form the category $\mathsf{2Sig}$.

We now study the existence of colimits in $\mathsf{2Sig}$. We know that Sig is cocomplete, and we use this knowledge in our study of $\mathsf{2Sig}$, by relating the two categories:

Let $F_{\mathsf{Sig}} : \mathsf{Sig} \to \mathsf{2Sig}$ be the functor which associates to any 1-signature Σ the empty family of equations, $F_{\mathsf{Sig}}(\Sigma) := (\Sigma, \emptyset)$. Call $U_{\mathsf{Sig}} : \mathsf{2Sig} \to \mathsf{Sig}$ the forgetful functor defined on objects as $U_{\mathsf{Sig}}(\Sigma, E) := \Sigma$.

▶ **Lemma 21** (`TwoSig_OneSig_is_right_adjoint`, `OneSig_TwoSig_fully_faithful`). *We have* $F_{\mathsf{Sig}} \dashv U_{\mathsf{Sig}}$. *Furthermore,* U_{Sig} *is a coreflection.*

We are interested in specifying new languages by "gluing together" simpler ones. On the level of 2-signatures, this is done by taking the coproduct, or, more generally, the pushout of 2-signatures:

[7] As noticed by an anonymous referee, this definition of "modularity" does not seem related to the specific meaning it has in the rewriting community (see, for example, [14]).

▶ **Theorem 22** (TwoSig_PushoutsSET)**.** *The category* 2Sig *has pushouts.*

Coproducts are computed by taking the union of the equations and the coproducts of the underlying 1-signatures. Coequalizers are computed by keeping the equations of the codomain and taking the coequalizer of the underlying 1-signatures. Thus, by decomposing any colimit into coequalizers and coproducts, we have this more general result:

▶ **Proposition 23.** *The category* 2Sig *is cocomplete and* U_{Sig} *preserves colimits.*

We now turn to our modularity result, which states that the initial model of a coproduct of two 2-signatures is the coproduct of the initial models of each 2-signature. More generally, the two languages can be amalgamated along a common "core language", by considering a pushout rather than a coproduct.

For a precise statement of that result, we define a "total category of models of 2-signatures":

▶ **Definition 24.** *The category* $\int_{(\Sigma,E)} \mathsf{Mon}^{(\Sigma,E)}$*, or* 2Mod *for short, has, as objects, pairs* $((\Sigma, E), M)$ *of a 2-signature* (Σ, E) *and a model* M *of* (Σ, E)*.*

A morphism from $((\Sigma_1, E_1), M_1)$ *to* $((\Sigma_2, E_2), M_2)$ *is a pair* (m, f) *consisting of a morphism* $m : (\Sigma_1, E_1) \to (\Sigma_2, E_2)$ *of 2-signatures and a morphism* $f : M_1 \to m^*M_2$ *of* (Σ_1, E_1)*-models (or, equivalently, of* Σ_1*-models).*

This category of models of 2-signatures contains the models of 1-signatures as a coreflective subcategory. Let $F_{\mathsf{Mod}} : \mathsf{Mod} \to 2\mathsf{Mod}$ be the functor which associates to any 1-model (Σ, M) the empty family of equations, $F_{\mathsf{Mod}}(\Sigma, M) := (F_{\mathsf{Sig}}(\Sigma), M)$. Conversely, the forgetful functor $U_{\mathsf{Mod}} : 2\mathsf{Mod} \to \mathsf{Mod}$ maps $((\Sigma, E), M)$ to (Σ, M).

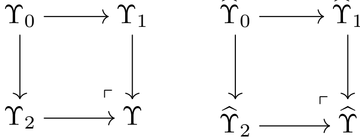
▶ **Lemma 25** (TwoMod_OneMod_is_right_adjoint, OneMod_TwoMod_fully_faithful)**.** *We have* $F_{\mathsf{Mod}} \dashv U_{\mathsf{Mod}}$*. Furthermore,* U_{Mod} *is a coreflection.*

The modularity result is a consequence of the following technical result:

▶ **Proposition 26** (TwoMod_cleaving)**.** *The forgetful functor* $2\pi : 2\mathsf{Mod} \to 2\mathsf{Sig}$ *is a Grothendieck fibration.*

The *modularity result* below is analogous to the modularity result for 1-signatures [2, Theorem 32]:

▶ **Theorem 27** (Modularity for 2-signatures, pushout_in_big_rep)**.** *Suppose we have a pushout diagram of effective 2-signatures, as on the left below. This pushout gives rise to a commutative square of morphisms of models in* 2Mod *as on the right below, where we only write the second components, omitting the (morphisms of) signatures. This square is a pushout square.*



Intuitively, the 2-signatures Υ_1 and Υ_2 specify two extensions of the 2-signature Υ_0, and Υ is the smallest extension containing both these extensions. By Theorem 27 the initial model of Υ is the "smallest model containing both the languages generated by Υ_1 and Υ_2".

4.4 Initial Semantics for 2-Signatures

We now turn to the problem of constructing the initial model of a 2-signature (Σ, E). More specifically, we identify sufficient conditions for (Σ, E) to admit an initial object $\widehat{(\Sigma, E)}$ in the category of models. Our approach is very straightforward: we seek to construct $\widehat{(\Sigma, E)}$ by applying a suitable quotient construction to the initial object $\hat{\Sigma}$ of Mon^{Σ}.

This leads immediately to our first requirement on (Σ, E), which is that Σ must be an effective 1-signature. (For instance, we can assume that Σ is an algebraic 1-signature, see Theorem 2.) This is a very natural hypothesis, since in the case where E is the empty family of Σ-equations, it is obviously a necessary and sufficient condition.

Some Σ-equations are never satisfied. In that case, the category $\mathsf{Mon}^{(\Sigma,E)}$ is empty. For example, given any 1-signature Σ, consider the Σ-equation $\mathsf{inl}, \mathsf{inr} : \Theta \rightrightarrows \Theta + \Theta$ given by the left and right inclusion. This is obviously an unsatisfiable Σ-equation. We have to find suitable hypotheses to rule out such unsatisfiable Σ-equations. This motivates the notion of *elementary* equations.

▶ **Definition 28.** *Given a 1-signature Σ, a Σ-module S is **nice** if S sends pointwise epimorphic Σ-model morphisms to pointwise epimorphic module morphisms.*

▶ **Definition 29** (`elementary_equation`). *Given a 1-signature Σ, an **elementary Σ-equation** is a Σ-equation such that*

- *the target is a finite derivative of the tautological 2-signature Θ, i.e., of the form $\Theta^{(n)}$ for some $n \in \mathbb{N}$, and*
- *the source is a nice Σ-module.*

▶ **Example 30** (`BindingSigAreEpiSig`). Any algebraic 1-signature is nice [2, Example 45]. Thus, any Σ-equation between an algebraic 1-signature and $\Theta^{(n)}$, for some natural number n, is elementary.

▶ **Definition 31.** *A 2-signature (Σ, E) is said **algebraic** if Σ is algebraic and E is a family of elementary equations.*

▶ **Theorem 32** (`elementary_equations_on_alg_preserve_initiality`). *Any algebraic 2-signature has an initial model.*

The proof of Theorem 32 is given in Section 5.

▶ **Example 33.** The 2-signature of lambda calculus modulo β and η equations given in Example 14 is algebraic. Its initial model is precisely the monad $\mathsf{LC}_{\beta\eta}$ of lambda calculus modulo $\beta\eta$ equations.

The instantiation of the formalized Theorem 32 to this 2-signature is done in `LCBetaEta`[8].

Let us mention finally that, using the axiom of choice, we can take a similar quotient on all the 1-models of Σ:

▶ **Proposition 34** (`ModEq_Mod_is_right_adjoint`, `ModEq_Mod_fully_faithful`). *Here we assume the axiom of choice. The forgetful functor from the category $\mathsf{Mon}^{(\Sigma,E)}$ of 2-models of (Σ, E) to the category Mon^{Σ} of Σ-models has a left adjoint. Moreover, the left adjoint is a reflector.*

[8] An initiality result for this particular case was also previously discussed and proved formally in the Coq proof assistant in [17].

5 Proof of Theorem 32

Our main technical result on effectiveness is the following Lemma 35. In Theorem 32, we give a much simpler criterion that encompasses all the examples we give.

The main technical result is encapsulated in the following lemma.

▶ **Lemma 35** (`elementary_equations_preserve_initiality`). *Let (Σ, E) be a 2-signature such that:*

1. *Σ sends epimorphic natural transformations to epimorphic natural transformations,*
2. *E is a family of elementary equations,*
3. *the initial 1-model of Σ exists,*
4. *the initial 1-model of Σ preserves epimorphisms,*
5. *the image by Σ of the initial 1-model of Σ preserves epimorphisms.*

Then, the category of 2-models of (Σ, E) has an initial object.

Before tackling the proof of Lemma 35, we discuss how to derive Theorem 32 from it, and we prove some auxiliary results.

We start with a lemma about preservation of epimorphisms:

▶ **Lemma 36** (`algebraic_model_Epi` and `BindingSig_on_model_isEpi`). *Let Σ be an algebraic 1-signature. Then $\hat{\Sigma}$ and $\Sigma(\hat{\Sigma})$ preserve epimorphisms.*

Now we have everything we need to prove Theorem 32:

Proof of Theorem 32. The "epimorphism" hypotheses of Lemma 35 are used to transfer structure from the initial model $\hat{\Sigma}$ of the 1-signature Σ onto a suitable quotient. There are different ways to prove these hypotheses:

- The axiom of choice implies conditions 4 and 5 since, in this case, any epimorphism in Set is split and thus preserved by any functor.
- Condition 5 is a consequence of condition 4 if Σ sends monads preserving epimorphisms to modules preserving epimorphisms.
- If Σ is algebraic, then conditions 1, 3, 4 and 5 are satisfied, cf. Example 30 and Lemma 36.

From the remarks above, we derive the simpler and weaker statement of Theorem 32 that covers all our examples, which are algebraic. ◀

The rest of this section is dedicated to the proof of the main technical result, Lemma 35. The reader inclined to do so may safely skip this part, and rely on the correctness of the machine-checked proof instead.

The proof of Lemma 35 uses some quotient constructions that we present now:

▶ **Proposition 37** (`u_monad_def`). *Given a monad R preserving epimorphisms and a collection of monad morphisms $(f_i : R \to S_i)_{i \in I}$, there exists a* quotient *monad $R/(f_i)$ together with a* projection *$p^R \colon R \longrightarrow R/(f_i)$, which is a morphism of monads such that each f_i factors through p.*

Proof. The set $R/(f_i)(X)$ is computed as the quotient of $R(X)$ with respect to the relation $x \sim y$ if and only if $f_i(x) = f_i(y)$ for each $i \in I$. This is a straightforward adaptation of Lemma 47 of [2]. ◀

Note that the epimorphism preservation is implied by the axiom of choice, but can be proven for the monad underlying the initial model $\hat{\Sigma}$ of an algebraic 1-signature Σ even without resorting to the axiom of choice.

The above construction can be transported on Σ-models:

▶ **Proposition 38** (u_rep_def). *Let Σ be a 1-signature sending epimorphic natural transformations to epimorphic natural transformations, and let R be a Σ-model such that R and $\Sigma(R)$ preserve epimorphisms. Let $(f_i : R \to S_i)_{i \in I}$ be a collection of Σ-model morphisms. Then the monad $R/(f_i)$ has a natural structure of Σ-model and the quotient map $p^R \colon R \longrightarrow R/(f_i)$ is a morphism of Σ-models. Any morphism f_i factors through p^R in the category of Σ-models.*

The fact that R and $\Sigma(R)$ preserve epimorphisms is implied by the axiom of choice. The proof follows the same line of reasoning as the proof of Proposition 37.

Now we are ready to prove the main technical lemma:

Proof of Lemma 35. Let Σ be an effective 1-signature, and let E be a set of elementary Σ-equations. The plan of the proof is as follows:

1. Start with the initial model $(\hat{\Sigma}, \sigma)$, with $\sigma : \Sigma(\hat{\Sigma}) \to \hat{\Sigma}$.
2. Construct the quotient model $\hat{\Sigma}/(f_i)$ according to Proposition 38 where $(f_i : \hat{\Sigma} \to S_i)_i$ is the collection of all initial Σ-morphisms from $\hat{\Sigma}$ to any Σ-model satisfying the equations. We denote by $\sigma/(f_i) : \Sigma(\hat{\Sigma}/(f_i)) \to \hat{\Sigma}/(f_i)$ the action of the quotient model.
3. Given a model M of the 2-signature (Σ, E), we obtain a morphism $i_M : \hat{\Sigma}/(f_i) \to M$ from Proposition 38. Uniqueness of i_M is shown using epimorphicity of the projection $p : \hat{\Sigma} \to \hat{\Sigma}/(f_i)$. For this, it suffices to show uniqueness of the composition $i_M \circ p : \hat{\Sigma} \to M$ in the category of 1-models of Σ, which follows from initiality of $\hat{\Sigma}$.
4. The verification that $\big(\hat{\Sigma}/(f_i), \sigma/(f_i)\big)$ satisfies the equations is given below. Actually, it follows the same line of reasoning as in the proof of Proposition 37 that $\hat{\Sigma}/(f_i)$ satisfies the monad equations.

Let $e = (e_1, e_2) : U \to \Theta^{(n)}$ be an elementary equation of E. We want to prove that the two arrows

$$e_{1,\hat{\Sigma}/(f_i)}, e_{2,\hat{\Sigma}/(f_i)} \colon U(\hat{\Sigma}/(f_i)) \longrightarrow (\hat{\Sigma}/(f_i))^{(n)}$$

are equal. As p is an epimorphic natural transformation, $U(p)$ also is by definition of an elementary equation. It is thus sufficient to prove that

$$e_{1,\hat{\Sigma}/(f_i)} \circ U(p) = e_{2,\hat{\Sigma}/(f_i)} \circ U(p) \ ,$$

which, by naturality of e_1 and e_2, is equivalent to $p^{(n)} \circ e_{1,\hat{\Sigma}} = p^{(n)} \circ e_{2,\hat{\Sigma}}$.

Let x be an element of $U(\hat{\Sigma})$ and let us show that $p^{(n)}(e_{1,\hat{\Sigma}}(x)) = p^{(n)}(e_{2,\hat{\Sigma}}(x))$. By definition of $\hat{\Sigma}/(f_i)$ as a pointwise quotient (see Proposition 37), it is enough to show that for any j, the equality $f_j^{(n)}(e_{1,\hat{\Sigma}}(x)) = f_j^{(n)}(e_{2,\hat{\Sigma}}(x))$ is satisfied. Now, by naturality of e_1 and e_2, this equation is equivalent to $e_{1,S_j}(U(f_j)(x))) = e_{2,S_j}(U(f_j)(x)))$ which is true since S_j satisfies the equation $e_1 = e_2$. ◀

6 Examples of algebraic 2-signatures

We already illustrated our theory by looking at the paradigmatic case of lambda calculus modulo β- and η-equations (Examples 11 and 33). This section collects further examples of application of our results.

In our framework, complex signatures can be built out of simpler ones by taking their coproducts. Note that the class of algebraic 2-signatures encompasses the algebraic 1-signatures and is closed under arbitrary coproducts: the prototypical examples of algebraic 2-signatures given in this section can be combined with any other algebraic 2-signature, yielding an effective 2-signature thanks to Theorem 32.

6.1 Monoids

We begin with an example of monad for a first-order syntax with equations. Given a set X, we denote by $M(X)$ the free monoid built over X. This is a classical example of monad over the category of (small) sets. The monoid structure gives us, for each set X, two maps $m_X\colon M(X) \times M(X) \longrightarrow M(X)$ and $e_X\colon 1 \longrightarrow M(X)$ given by the product and the identity respectively. It can be easily verified that $m\colon M^2 \longrightarrow M$ and $e\colon 1 \longrightarrow M$ are M-module morphisms. In other words, $(M, \rho) = (M, [m, e])$ is a model of the 1-signature $\Sigma = \Theta \times \Theta + 1$.

We break the tautological morphism of Σ-modules (cf. Example 6) into constituent pieces, defining $\mathsf{m} := \tau \circ \mathsf{inl} : \Theta \times \Theta \to \Theta$ and $\mathsf{e} := \tau \circ \mathsf{inr} : 1 \to \Theta$.

Over the 1-signature Σ we specify equations postulating *associativity* and *left and right unitality* as follows:

$$\begin{array}{l} \Theta^3 \xrightarrow{\Theta\times\mathsf{m}} \Theta^2 \xrightarrow{\mathsf{m}} \Theta \\ \Theta^3 \xrightarrow[\mathsf{m}\times\Theta]{} \Theta^2 \xrightarrow[\mathsf{m}]{} \Theta \end{array} \qquad \begin{array}{l} \Theta \xrightarrow{\mathsf{e}\times\Theta} \Theta^2 \xrightarrow{\mathsf{m}} \Theta \\ \Theta \xrightarrow[1]{} \Theta \end{array} \qquad \begin{array}{l} \Theta \xrightarrow{\Theta\times\mathsf{e}} \Theta^2 \xrightarrow{\mathsf{m}} \Theta \\ \Theta \xrightarrow[1]{} \Theta \end{array}$$

and we denote by E the family consisting of these three Σ-equations. All are elementary since their codomain is Θ, and their domain a product of Θs.

One checks easily that $(M, [m, e])$ is the initial model of (Σ, E).

Several other classical (equational) algebraic theories, such as groups and rings, can be treated similarly, see Section 6.3 below. However, at the present state we cannot model theories with partial construction (e.g., fields).

6.2 Colimits of algebraic 2-signatures

In this section, we argue that our framework encompasses any colimit of algebraic 2-signatures.

Actually, the class of algebraic 2-signatures is not stable under colimits, as this is not even the case for algebraic 1-signatures. However, we can weaken this statement as follows:

▶ **Proposition 39.** *Given any colimit of algebraic 2-signatures, there is an algebraic 2-signature yielding an isomorphic category of models.*

Proof. As the class of algebraic 2-signatures is closed under arbitrary coproducts, using the decomposition of colimits into coproducts and coequalizers, any colimit Ξ of algebraic 2-signatures can be expressed as a coequalizer of two morphisms f, g between some algebraic 2-signatures (Σ_1, E_1) and (Σ_2, E_2),

$$(\Sigma_1, E_1) \overset{f}{\underset{g}{\rightrightarrows}} (\Sigma_2, E_2) \xrightarrow{p} \Xi = (\Sigma_3, E_2) .$$

where Σ_3 is the coequalizer of the 1-signatures morphisms f and g. Note that the set of equations of Ξ is E_2, by definition of the coequalizer in the category of 2-signatures. Now, consider the algebraic 2-signature $\Xi' = (\Sigma_2, E_2 + (2))$ consisting of the 1-signature Σ_2 and the equations of E_2 plus the following elementary equation (see Example 30):

$$\begin{array}{l} \Sigma_1 \xrightarrow{f} \Sigma_2 \xrightarrow{\tau^{\Sigma_2}} \Theta \\ \Sigma_1 \xrightarrow[g]{} \Sigma_2 \xrightarrow[\tau^{\Sigma_2}]{} \Theta \end{array} \tag{2}$$

We show that Mon^{Ξ} and $\mathsf{Mon}^{\Xi'}$ are isomorphic. A model of Ξ' is a monad R together with an R-module morphism $r : \Sigma_2(R) \to R$ such that $r \circ f_R = r \circ g_R$ and that the equations of E_2 are satisfied. By universal property of the coequalizer, this is exactly the same as giving an R-module morphism $\Sigma_3(R) \to R$ satisfying the equations of E_2, i.e., giving R an action of $\Xi = (\Sigma_3, E_2)$.

It is straightforward to check that this correspondence yields an isomorphism between the category of models of Ξ and the category of models of Ξ'. ◀

This proposition, together with the following corollary, allow us to recover all the examples presented in [2], as colimits of algebraic 1-signatures: syntactic commutative binary operator, maximum operator, application à la differential lambda calculus, syntactic closure operator, integrated substitution operator, coherent fixpoint operator.

▶ **Corollary 40.** *If F is a finitary endofunctor on* Set*, then there is an algebraic 2-signature whose category of models is isomorphic to the category of 1-models of the 1-signature $F \cdot \Theta$.*

Proof. It is enough to prove that $F \cdot \Theta$ is a colimit of algebraic 1-signatures.

As F is finitary, it is isomorphic to the coend $\int^{n\in\mathbb{N}} F(n) \times _^n$ where $\mathbb{N}$ is the full subcategory of Set of finite ordinals (see, e.g., [26, Example 3.19]). As colimits are computed pointwise, the 1-signature $F \cdot \Theta$ is the coend $\int^{n\in\mathbb{N}} F(n) \times \Theta^n$, and as such, it is a colimit of algebraic 2-signatures. ◀

However, we do not know whether we can recover our theorem [2, Theorem 35] stating that any presentable 1-signature is effective.

6.3 Algebraic theories

From the categorical point of view, several fundamental algebraic structures in mathematics can be conveniently and elegantly described using finitary monads. For instance, the category of monoids can be seen as the category of Eilenberg–Moore algebras of the monad of lists. Other important examples, like groups and rings, can be treated analogously. A classical reference on the subject is the work of Manes, where such monads are significantly called *finitary algebraic theories* [22, Definition 3.17].

We want to show that such "algebraic theories" fit in our framework, in the sense that they can be incorporated into an algebraic 2-signature, with the effect of enriching the initial model with the operations of the algebraic theory, subject to the axioms of the algebraic theory.

For a finitary monad T, Corollary 40 says how to encode the 1-signature $T \cdot \Theta$ as an algebraic 2-signature (Σ_T, E_T). Models are monads R together with an R-linear morphism $r : T \cdot R \to R$.

Now, for any model (R, m) of $T \cdot \Theta$, we would like to enforce the usual T-algebra equations on the action m. This is done thanks to the following equations, where τ denotes the tautological morphism of $T \cdot \Theta$-modules:

$$
\begin{array}{ll}
\Theta \xrightarrow{\eta_T \cdot \Theta} T \cdot \Theta \xrightarrow{\tau} \Theta & \qquad T \cdot T \cdot \Theta \xrightarrow{\mu_T \cdot \Theta} T \cdot \Theta \xrightarrow{\tau} \Theta \\
\Theta \xrightarrow[1]{\qquad\qquad} \Theta & \qquad T \cdot T \cdot \Theta \xrightarrow[T\tau]{} T \cdot \Theta \xrightarrow[\tau]{} \Theta
\end{array}
\qquad (3)
$$

The first equation is clearly elementary. The second one is elementary thanks to the following lemma:

▶ **Lemma 41.** *Let F be a finitary endofunctor on* Set*. Then F preserves epimorphisms.*

Proof. An anonymous referee remarked that this is a consequence of the axiom of choice, because then any epimorphism in the category of Set is split, and thus preserved by any functor. Here we provide an alternative proof which does not rely on the axiom of choice. (However, it may require the excluded middle, depending on the chosen definition of finitary functor.)

As F is finitary, it is isomorphic to the coend $\int^{n\in\mathbb{N}} F(n) \times _^n$ [26, Example 3.19]. By decomposing it as a coequalizer of coproducts, we get an epimorphism $\alpha : \coprod_{n\in\mathbb{N}} F(n) \times _^n \to F$. Now, let $f : X \to Y$ be a surjective function between two sets. We show that $F(f)$ is epimorphic. By naturality, the following diagram commutes:

$$\begin{array}{ccc} \coprod_{n\in\mathbb{N}} F(n) \times X^n & \xrightarrow{F(n)\times f^n} & \coprod_{n\in\mathbb{N}} F(n) \times Y^n \\ \downarrow{\scriptstyle \alpha_X} & & \downarrow{\scriptstyle \alpha_Y} \\ F(X) & \xrightarrow[F(f)]{} & F(Y) \end{array}$$

The composition along the top-right is epimorphic by composition of epimorphisms. Thus, the bottom left is also epimorphic, and so is $F(f)$ as the last morphism of this composition. ◀

In conclusion, we have exhibited the algebraic 2-signature (Σ_T, E'_T), where E'_T extends the family E_T with the two elementary equations of Diagram 3. This signature allows to enrich any other algebraic 2-signature with the operations of the algebraic theory T, subject to the relevant equations.

6.4 Fixpoint operator

Here, we show the algebraic 2-signature corresponding to a fixpoint operator. In [2, Section 8.4] we studied fixpoint operators in the context of 1-signatures. In that setting, we treated a *syntactic* fixpoint operator called *coherent* fixpoint operator, somehow reminiscent of mutual letrec. We were able to impose many natural equations to this operator but we were not able to enforce the fixpoint equation. In this section, we show how a fixpoint operator can be fully specified by an algebraic 2-signature. We restrict our discussion to the unary case; the coherent family of multi-ary fixpoint operators presented in [2, Section 8.4], now including the fixpoint equations, can also be specified, in an analogous way, via an algebraic 2-signature.

Let us start by recalling the following

▶ **Definition 42.** *A **unary fixpoint operator for a monad R** [2, Definition 40] is a module morphism f from R' to R that makes the following diagram commute, where σ is the substitution morphism defined as the uncurrying (see Definition 10) of the identity morphism on Θ':*

$$\begin{array}{ccc} R' & \xrightarrow{(id_{R'},f)} & R' \times R \\ & {\scriptstyle f}\searrow \quad \swarrow{\scriptstyle \sigma_R} & \\ & R & \end{array}$$

In order to rephrase this definition, we introduce the obviously algebraic 2-signature Υ_{fix} consisting of the 1-signature $\Sigma_{\mathsf{fix}} = \Theta'$ and the family E_{fix} consisting of the single following Σ_{fix}-equation:

$$e_{\mathsf{fix}} : \begin{array}{l} \Theta' \xrightarrow{\langle 1,\tau\rangle} \Theta' \times \Theta \xrightarrow{\sigma} \Theta \\ \Theta' \xrightarrow[\tau]{} \Theta \end{array} \tag{4}$$

This allows us to rephrase the previous definition as follows: a unary fixpoint operator for a monad R is just an action of the 2-signature Υ_{fix} in R.

The name "fixpoint operator" is motivated by the following proposition:

▶ **Proposition 43** ([2, Proposition 41]). *Fixpoint combinators are in one-to-one correspondence with actions of Υ_{fix} in the monad $\mathsf{LC}_{\beta\eta}$ of the lambda calculus modulo β- and η-equality.*

Recall that fixpoint combinators are lambda terms Y satisfying, for any (possibly open) term t, the equation

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

Explicitly, such a combinator Y induces a fixpoint operator $\hat{Y} : \mathsf{LC}'_{\beta\eta} \to \mathsf{LC}_{\beta\eta}$ which associates, to any term t depending on an additional variable $*$, the term $\hat{Y}(t) := \mathsf{app}(Y, \mathsf{abs}\ t)$.

7 Recursion

In this section, we explain how a recursion principle can be derived from our initiality result, and give an example of a morphism – a *translation* – between monads defined via the recursion principle.

7.1 Principle of recursion

In our context, the recursion principle is a recipe for constructing a morphism from the monad underlying the initial model of a 2-signature to an arbitrary monad.

▶ **Proposition 44** (Recursion principle). *Let S be the monad underlying the initial model of the 2-signature Υ. To any action a of Υ in T is associated a monad morphism $\hat{a} : S \to T$.*

Proof. The action a defines a 2-model M of Υ, and $\hat{a}$ is the monad morphism underlying the initial morphism to M. ◀

Hence the recipe consists in the following two steps:

1. give T an action of the 1-signature Σ;
2. check that all the equations in E are satisfied for the induced model.

In the next section, we illustrate this principle.

7.2 Translation of lambda calculus with fixpoint to lambda calculus

In this section, we consider the 2-signature $\Upsilon_{\mathsf{LC}_{\beta\eta,\mathsf{fix}}} := \Upsilon_{\mathsf{LC}_{\beta\eta}} + \Upsilon_{\mathsf{fix}}$ where the two components have been introduced above (see Example 18 and Section 6.4).

As a coproduct of algebraic 2-signatures, $\Upsilon_{\mathsf{LC}_{\beta\eta,\mathsf{fix}}}$ is itself algebraic, and thus the initial model exists. The underlying monad $\mathsf{LC}_{\beta\eta,\mathsf{fix}}$ of the initial model can be understood as the monad of lambda calculus modulo β and η enriched with an *explicit* fixpoint operator $\mathsf{fix} : \mathsf{LC}'_{\beta\eta,\mathsf{fix}} \longrightarrow \mathsf{LC}_{\beta\eta,\mathsf{fix}}$. Now we build by recursion a monad morphism from this monad to the "bare" monad $\mathsf{LC}_{\beta\eta}$ of lambda calculus modulo β and η.

As explained in Section 7.1, we need to define an action of $\Upsilon_{\mathsf{LC}_{\beta\eta,\mathsf{fix}}}$ in $\mathsf{LC}_{\beta\eta}$, that is to say an action of $\Upsilon_{\mathsf{LC}_{\beta\eta}}$ plus an action of Υ_{fix}. For the action of $\Upsilon_{\mathsf{LC}_{\beta\eta}}$, we take the one yielding the initial model.

Now, in order to find an action of Υ_{fix} in $\mathsf{LC}_{\beta\eta}$, we choose a fixpoint combinator Y (say the one of Curry) and take the action $\hat{Y}$ as defined at the end of Section 6.4.

In more concrete terms, our translation is a kind of compilation which replaces each occurrence of the explicit fixpoint operator $\mathsf{fix}(t)$ with $\mathsf{app}(Y, \mathsf{abs}\ t)$.

References

1. Benedikt Ahrens. Modules over relative monads for syntax and semantics. *Mathematical Structures in Computer Science*, 26:3–37, 2016. doi:10.1017/S0960129514000103.
2. Benedikt Ahrens, André Hirschowitz, Ambroise Lafont, and Marco Maggesi. High-Level Signatures and Initial Semantics. In Dan Ghica and Achim Jung, editors, *27th EACSL Annual Conference on Computer Science Logic (CSL 2018)*, volume 119 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:22, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CSL.2018.4.
3. Michael Barr. Coequalizers and free triples. *Mathematische Zeitschrift*, 116(4):307–322, December 1970. doi:10.1007/BF01111838.
4. Ranald Clouston. Binding in Nominal Equational Logic. *Electr. Notes Theor. Comput. Sci.*, 265:259–276, 2010. doi:10.1016/j.entcs.2010.08.016.
5. Maribel Fernández and Murdoch J. Gabbay. Closed nominal rewriting and efficiently computable nominal algebra equality. In Karl Crary and Marino Miculan, editors, *Proceedings 5th International Workshop on Logical Frameworks and Meta-languages: Theory and Practice, LFMTP 2010, Edinburgh, UK, 14th July 2010.*, volume 34 of *EPTCS*, pages 37–51, 2010. doi:10.4204/EPTCS.34.5.
6. Marcelo P. Fiore and Makoto Hamana. Multiversal Polymorphic Algebraic Theories: Syntax, Semantics, Translations, and Equational Logic. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 520–529. IEEE Computer Society, 2013. doi:10.1109/LICS.2013.59.
7. Marcelo P. Fiore and Chung-Kil Hur. On the construction of free algebras for equational systems. *Theor. Comput. Sci.*, 410(18):1704–1729, 2009. doi:10.1016/j.tcs.2008.12.052.
8. Marcelo P. Fiore and Chung-Kil Hur. Second-Order Equational Logic (Extended Abstract). In Anuj Dawar and Helmut Veith, editors, *CSL*, volume 6247 of *Lecture Notes in Computer Science*, pages 320–335. Springer, 2010. doi:10.1007/978-3-642-15205-4_26.
9. Marcelo P. Fiore and Ola Mahmoud. Second-Order Algebraic Theories (Extended Abstract). In Petr Hlinený and Antonín Kucera, editors, *MFCS*, volume 6281 of *Lecture Notes in Computer Science*, pages 368–380. Springer, 2010. doi:10.1007/978-3-642-15155-2_33.
10. Marcelo P. Fiore, Gordon D. Plotkin, and Daniele Turi. Abstract Syntax and Variable Binding. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, pages 193–202, 1999. doi:10.1109/LICS.1999.782615.
11. Marcelo P. Fiore and Philip Saville. List Objects with Algebraic Structure. In Dale Miller, editor, *2nd International Conference on Formal Structures for Computation and Deduction (FSCD 2017)*, volume 84 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:18, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.FSCD.2017.16.
12. Murdoch J. Gabbay and Andrew M. Pitts. A New Approach to Abstract Syntax Involving Binders. In *14th Annual Symposium on Logic in Computer Science*, pages 214–224, Washington, DC, USA, 1999. IEEE Computer Society Press. doi:10.1109/LICS.1999.782617.
13. Neil Ghani, Tarmo Uustalu, and Makoto Hamana. Explicit substitutions and higher-order syntax. *Higher-Order and Symbolic Computation*, 19(2-3):263–282, 2006. doi:10.1007/s10990-006-8748-4.
14. Bernhard Gramlich. Modularity in term rewriting revisited. *Theoretical Computer Science*, 464:3–19, 2012. New Directions in Rewriting (Honoring the 60th Birthday of Yoshihito Toyama). doi:10.1016/j.tcs.2012.09.008.
15. Makoto Hamana. Term Rewriting with Variable Binding: An Initial Algebra Approach. In *Proceedings of the 5th ACM SIGPLAN International Conference on Principles and Practice of Declaritive Programming*, PPDP '03, pages 148–159, New York, NY, USA, 2003. ACM. doi:10.1145/888251.888266.

16 André Hirschowitz and Marco Maggesi. Modules over Monads and Linearity. In D. Leivant and R. J. G. B. de Queiroz, editors, *WoLLIC*, volume 4576 of *Lecture Notes in Computer Science*, pages 218–237. Springer, 2007. `doi:10.1007/978-3-540-73445-1_16`.

17 André Hirschowitz and Marco Maggesi. Modules over monads and initial semantics. *Information and Computation*, 208(5):545–564, May 2010. Special Issue: 14th Workshop on Logic, Language, Information and Computation (WoLLIC 2007). `doi:10.1016/j.ic.2009.07.003`.

18 Martin Hofmann. Semantical Analysis of Higher-Order Abstract Syntax. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, pages 204–213. IEEE Computer Society, 1999. `doi:10.1109/LICS.1999.782616`.

19 G. Maxwell Kelly and A. John Power. Adjunctions whose counits are coequalizers, and presentations of finitary enriched monads. *Journal of Pure and Applied Algebra*, 89(1):163–179, 1993. `doi:10.1016/0022-4049(93)90092-8`.

20 Alexander Kurz and Daniela Petrisan. On universal algebra over nominal sets. *Mathematical Structures in Computer Science*, 20(2):285–318, 2010. `doi:10.1017/S0960129509990399`.

21 Christoph Lüth and Neil Ghani. Monads and Modular Term Rewriting. In Eugenio Moggi and Giuseppe Rosolini, editors, *Category Theory and Computer Science, 7th International Conference, CTCS '97*, volume 1290 of *Lecture Notes in Computer Science*, pages 69–86. Springer, 1997. `doi:10.1007/BFb0026982`.

22 Ernest Manes. *Algebraic Theories*, volume 26 of *Graduate Texts in Mathematics*. Springer, 1976.

23 The Coq development team. *The Coq Proof Assistant, version 8.9*, 2019. Version 8.9. URL: `http://coq.inria.fr`.

24 A. John Power. Abstract Syntax: Substitution and Binders: Invited Address. *Electr. Notes Theor. Comput. Sci.*, 173:3–16, 2007. `doi:10.1016/j.entcs.2007.02.024`.

25 Sam Staton. An Algebraic Presentation of Predicate Logic (Extended Abstract). In Frank Pfenning, editor, *Foundations of Software Science and Computation Structures - 16th International Conference, FOSSACS 2013*, volume 7794 of *Lecture Notes in Computer Science*, pages 401–417. Springer, 2013. `doi:10.1007/978-3-642-37075-5_26`.

26 Jiří Velebil and Alexander Kurz. Equational presentations of functors and monads. *Mathematical Structures in Computer Science*, 21(2):363–381, 2011. `doi:10.1017/S0960129510000575`.

27 Vladimir Voevodsky, Benedikt Ahrens, Daniel Grayson, et al. UniMath — a computer-checked library of univalent mathematics. Available at `https://github.com/UniMath/UniMath`.

Towards the Average-Case Analysis of Substitution Resolution in λ-Calculus

Maciej Bendkowski
Jagiellonian University, Faculty of Mathematics and Computer Science,
Theoretical Computer Science Department, ul. Prof. Łojasiewicza 6, 30–348 Kraków, Poland
maciej.bendkowski@tcs.uj.edu.pl

Abstract

Substitution resolution supports the computational character of β-reduction, complementing its execution with a capture-avoiding exchange of terms for bound variables. Alas, the meta-level definition of substitution, masking a non-trivial computation, turns β-reduction into an atomic rewriting rule, despite its varying operational complexity. In the current paper we propose a somewhat indirect average-case analysis of substitution resolution in the classic λ-calculus, based on the quantitative analysis of substitution in $\lambda\upsilon$, an extension of λ-calculus internalising the υ-calculus of explicit substitutions. Within this framework, we show that for any fixed $n \geq 0$, the probability that a uniformly random, conditioned on size, $\lambda\upsilon$-term υ-normalises in n normal-order (i.e. leftmost-outermost) reduction steps tends to a computable limit as the term size tends to infinity. For that purpose, we establish an effective hierarchy $(\mathcal{G}_n)_n$ of regular tree grammars partitioning υ-normalisable terms into classes of terms normalising in n normal-order rewriting steps. The main technical ingredient in our construction is an inductive approach to the construction of $\mathcal{G}_{n+1}$ out of $\mathcal{G}_n$ based, in turn, on the algorithmic construction of finite intersection partitions, inspired by Robinson's unification algorithm. Finally, we briefly discuss applications of our approach to other term rewriting systems, focusing on two closely related formalisms, i.e. the full $\lambda\upsilon$-calculus and combinatory logic.

2012 ACM Subject Classification Theory of computation → Lambda calculus; Mathematics of computing → Generating functions

Keywords and phrases lambda calculus, explicit substitutions, complexity, combinatorics

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.7

Funding *Maciej Bendkowski*: Maciej Bendkowski was partially supported within the Polish National Science Center grant 2016/21/N/ST6/01032.

1 Introduction

Traditional, machine-based computational models, such as Turing machines or RAMs, admit a natural notion of an atomic computation step, closely reflecting the actual operational cost of executing the represented computations. Unfortunately, this is not quite the case for computational models based on term rewriting systems with substitution, such as the classic λ-calculus. Given the (traditionally) epitheoretic nature of substitution, the single rewriting rule of β-reduction $(\lambda x.a)b \to_\beta a[x := b]$ masks a non-trivial computation of resolving (i.e. executing) the pending substitution of b for occurrences of x in a. Moreover, unlike machine-based models, λ-calculus (as other term rewriting systems) does not impose a strict, deterministic evaluation mechanism. Consequently, various strategies for resolving substitutions can be used, even more obfuscating the operational semantics of β-reduction and hence also its operational cost. Those subtle nuances hidden behind the implementation details of substitution resolution are in fact one of the core issues in establishing reasonable cost models for the classic λ-calculus, relating it with other, machine-based computational models, see [14].

© Maciej Bendkowski;
licensed under Creative Commons License CC-BY

In order to resolve this apparent inadequacy, Abadi et al. proposed to refine substitution in the classic λ-calculus and decompose it into a series of atomic rewriting steps, internalising in effect the calculus of executing substitutions [1]. Substitutions become first-class citizens and so can be manipulated together with regular terms. Consequently, the general framework of explicit substitutions provides a machine-independent setup for the operational semantics of substitution, based on a finite set of unit rewriting primitives. Remarkably, with the help of linear substitution calculus (a resource aware calculus of explicit substitutions) Accattoli and Dal Lago showed recently that the leftmost-outermost β-reduction strategy is a reasonable invariant cost model for λ-calculus, and hence it is able to simulate RAMs (or equivalent, machine-based models) within a polynomial time overhead [2].

Various subtleties of substitution resolution, reflected in the variety of available calculi of explicit substitutions, induce different operational semantics for executing substitutions in λ-calculus. This abundance of approaches is perhaps the main barrier in establishing a systematic, quantitative analysis of the operational complexity of substitution resolution and, among other things, a term rewriting analogue of classic average-case complexity analysis. In the current paper we propose a step towards filling this gap by offering a quantitative approach to substitution resolution in Lescanne's $\lambda\upsilon$-calculus of explicit substitutions [15]. In particular, we focus on the following, average-case analysis type of question. Having fixed arbitrary non-negative n, what is the probability that a (uniformly) random $\lambda\upsilon$-term of given size is υ-normalisable (i.e. can be reduced to a normal form without explicit substitutions) in exactly n leftmost-outermost reduction steps? Furthermore, how does this probability distribution change when the term size tends to infinity?

We address the above questions using a two-step approach. First, we exhibit an effective (i.e. computable) hierarchy $(\mathcal{G}_n)_n$ of unambiguous regular tree grammars with the property that $\mathcal{G}_n$ describes the language of terms υ-normalising in precisely n leftmost-outermost υ-rewriting steps. Next, borrowing techniques from analytic combinatorics, we analyse the limit proportion of terms υ-normalising in n normal-order steps. To that end, we construct appropriate generating functions and provide asymptotic estimates for the number of $\lambda\upsilon$-terms υ-normalising in n normal-order reduction steps. As a result, we base our approach on a direct quantitative analysis of the υ term rewriting system, measuring the operational cost of evaluating substitution in terms of the number of leftmost-outermost rewriting steps required to reach a (υ-)normal form.

The paper is structured as follows. In Section 2 we outline $\lambda\upsilon$-calculus and the framework of regular tree grammars, establishing the necessary terminology for the remainder of the paper. Next, in Section 3, we prepare the background for the construction of $(\mathcal{G}_n)_n$. In particular, we sketch its general, intuitive scheme. In Section 4 we introduce the main tool of finite intersection partitions and show that it is indeed constructible in the context of generated reduction grammars. Afterwards, in Section 5, we show how finite intersection partitions can be used in the construction of new productions in $\mathcal{G}_{n+1}$ based on productions in the grammar $\mathcal{G}_n$. Having constructed $(\mathcal{G}_n)_n$ we then proceed to the main quantitative analysis of υ-calculus using methods of analytic combinatorics, see Section 6. Finally, in Section 7 we discuss broader applications of our technique to other term rewriting systems, based on the examples of $\lambda\upsilon$-calculus and combinatory logic, and conclude the paper in the final Section 8.

2 Preliminaries

2.1 Lambda upsilon calculus

$\lambda\upsilon$ (lambda upsilon) is a simple, first-order term rewriting system extending the classic λ-calculus based on de Bruijn indices [11] with the calculus of resolving pending substitutions [15, 16]. Its formal terms, so-called $\lambda\upsilon$-terms, are comprised of de Bruijn indices $\underline{n}$,

application, abstraction, together with an additional, explicit *closure* operator $[\cdot]$ standing for unresolved substitutions. De Bruijn indices are represented in unary base expansion. In other words, $\underline{\mathsf{n}}$ is encoded as an n-fold application of the successor operator S to zero $\underline{\mathsf{0}}$. Substitutions, in turn, consist of three primitives, i.e. a constant *shift* $\uparrow$, a unary *lift* operator $\Uparrow$, mapping substitutions onto substitutions, and a unary *slash* operator $/$, mapping terms onto substitutions. Terms containing closures are called *impure* whereas terms without them are said to be *pure*. Figure 1 summarises the formal specification of $\lambda\upsilon$-terms and the corresponding rewriting system $\lambda\upsilon$.

$$\begin{aligned} t &::= \underline{\mathsf{n}} \mid \lambda t \mid tt \mid t[s] \\ s &::= t/ \mid \Uparrow(s) \mid \uparrow \\ \underline{\mathsf{n}} &::= \underline{\mathsf{0}} \mid \mathsf{S}\,\underline{\mathsf{n}}. \end{aligned}$$

(a) Terms of $\lambda\upsilon$-calculus.

$$\begin{aligned} (\lambda a)b &\to a[b/] && \text{(Beta)} \\ (ab)[s] &\to a[s](b[s]) && \text{(App)} \\ (\lambda a)[s] &\to \lambda(a[\Uparrow(s)]) && \text{(Lambda)} \\ \underline{\mathsf{0}}[a/] &\to a && \text{(FVar)} \\ (\mathsf{S}\,\underline{\mathsf{n}})[a/] &\to \underline{\mathsf{n}} && \text{(RVar)} \\ \underline{\mathsf{0}}[\Uparrow(s)] &\to \underline{\mathsf{0}} && \text{(FVarLift)} \\ (\mathsf{S}\,\underline{\mathsf{n}})[\Uparrow(s)] &\to \underline{\mathsf{n}}[s][\uparrow] && \text{(RVarLift)} \\ \underline{\mathsf{n}}[\uparrow] &\to \mathsf{S}\,\underline{\mathsf{n}}. && \text{(VarShift)} \end{aligned}$$

(b) Rewriting rules.

Figure 1 The $\lambda\upsilon$-calculus rewriting system.

▶ **Example 2.1.** Note that the well-known combinator $K = \lambda xy.x$ is represented in the de Bruijn notation as $\lambda\lambda\underline{1}$. The reverse application term $\lambda xy.yx$, on the other hand, is represented as $\lambda\lambda\underline{01}$. Consequently, in a single β-reduction step, it holds $(\lambda\lambda\underline{01})\,K \to_\beta \lambda\,(\underline{0}K)$. In $\lambda\upsilon$-calculus, however, this single β-reduction is decomposed into a series of small rewriting steps governing both the β-reduction as well as the subsequent substitution resolution. For instance, we have

$$\begin{aligned} (\lambda\lambda\underline{01})\,K &\to (\lambda\underline{01})\,[K/] \to (\lambda\,(\underline{01})\,[\Uparrow(K/)]) \to \lambda\,(\underline{0}[\Uparrow(K/)])\,(\underline{1}[\Uparrow(K/)]) \\ &\to \lambda\,(\underline{0}\,(\underline{1}[\Uparrow(K/)])) \to \lambda\,(\underline{0}\,(\underline{0}[K/][\uparrow])) \to \lambda\,(\underline{0}\,(K[\uparrow]))\,. \end{aligned} \tag{1}$$

Furthermore,

$$\begin{aligned} K[\uparrow] = (\lambda\lambda\underline{1})\,[\uparrow] &\to \lambda\,((\lambda\underline{1})[\Uparrow(\uparrow)]) \to \lambda\lambda\,(\underline{1}[\Uparrow(\Uparrow(\uparrow))]) \\ &\to \lambda\lambda\,(\underline{0}[\Uparrow(\uparrow)][\uparrow]) \to \lambda\lambda\,(\underline{0}[\uparrow]) \\ &\to \lambda\lambda\underline{1} = K \end{aligned} \tag{2}$$

hence indeed $(\lambda\lambda\underline{01})\,K$ rewrites to $\lambda\,(\underline{0}K)$.

Let us notice that in the presence of the erasing (RVar) and duplicating (App) rewriting rules, not all reduction sequences starting with the same term have to be of equal length. Like in the classic λ-calculus, depending on the considered term, some rewriting strategies might be more efficient then others.

$\lambda\upsilon$ enjoys a series of pleasing properties. Most notably, $\lambda\upsilon$ is confluent, correctly implements β-reduction of the classic λ-calculus, and preserves strong normalisation of closed terms [3]. Moreover, the υ fragment, i.e. $\lambda\upsilon$ without the (Beta) rule, is terminating. In

other words, each $\lambda\upsilon$-term is υ-normalising as can be shown using, for instance, polynomial interpretations [9]. In the current paper we focus on the normal-order (i.e. leftmost-outermost) evaluation strategy of υ-reduction. For convenience, we assume the following notational conventions. We use lowercase letters $a, b, c, \ldots$ to denote arbitrary terms and s (with or without subscripts) to denote substitutions. Moreover, we write $a \downarrow_n$ to denote the fact that a normalises to its υ-normal form in n normal-order υ-reduction steps. Sometimes, for further convenience, we also simply state that t normalises in n steps, without specifying the assumed evaluation strategy nor the specific rewriting steps and normal form.

2.2 Regular tree languages

We base our main construction in the framework of regular tree languages. In what follows, we outline their basic characteristics and that of corresponding regular tree grammars, introducing necessary terminology. We refer the curious reader to [10, Chapter II] for a more detailed exposition.

▶ **Definition 2.2** (Ranked alphabet). *A* ranked alphabet $\mathcal{F}$ *is a finite set of* function symbols *endowed with a corresponding* arity *function* $\mathsf{arity}\colon \mathcal{F} \to \mathbb{N}$. *We use* $\mathcal{F}_n$ *to denote the set of function symbols of arity* n, *i.e. function symbols* $f \in \mathcal{F}$ *such that* $\mathsf{arity}(f) = n$. *Function symbols of arity zero are called* constants. *As a notational convention, we use lowercase letters* $f, g, h, \ldots$ *to denote arbitrary function symbols.*

▶ **Definition 2.3** (Terms). *Let* X *be a finite set of* variables. *Then, the set* $\mathcal{T}_{\mathcal{F}}(X)$ *of* terms over $\mathcal{F}$ *is defined inductively as follows:*

- $X, \mathcal{F}_0 \subset \mathcal{T}_{\mathcal{F}}(X)$;
- *If* $f \in \mathcal{F}_n$ *and* $\alpha_1, \ldots, \alpha_n \in \mathcal{T}_{\mathcal{F}}(X)$, *then* $f(\alpha_1, \ldots, \alpha_n) \in \mathcal{T}_{\mathcal{F}}(X)$.

Terms not containing variables, in other words elements of $\mathcal{T}_{\mathcal{F}}(\varnothing)$, *are called* ground terms.

As a notational convention, we use lowercase Greek letters $\alpha, \beta, \gamma, \ldots$ *to denote arbitrary terms. Whenever it is clear from the context, we use the word* term *both to refer to the above structures as well as to denote* $\lambda\upsilon$*-terms.*

▶ **Definition 2.4** (Regular tree grammars). *A* regular tree grammar $\mathcal{G} = (S, \mathcal{N}, \mathcal{F}, \mathcal{P})$ *is a tuple consisting of:*

- *an* axiom $S \in \mathcal{N}$;
- *a finite set* $\mathcal{N}$ *of* non-terminal symbols;
- *a ranked alphabet* $\mathcal{F}$ *of* terminal symbols *such that* $\mathcal{F} \cap \mathcal{N} = \varnothing$; *and*
- *a finite set* $\mathcal{P}$ *of* productions *in form of* $N \to \alpha$ *such that* $N \in \mathcal{N}$ *and* $\alpha \in \mathcal{T}_{\mathcal{F}}(\mathcal{N})$.

A production $N \to \alpha$ *is* self-referencing *if* N *occurs in* α. *Otherwise, if* N *does not occur in* α, *we say that* $n \to \alpha$ *is* regular. *As a notational convention, we use capital letters* $X, Y, Z \ldots$ *to denote arbitrary non-terminal symbols.*

▶ **Definition 2.5** (Derivation relation). *The* derivation relation $\to_{\mathcal{G}}$ *associated with the grammar* $\mathcal{G} = (S, \mathcal{N}, \mathcal{F}, \mathcal{P})$ *is a relation on pairs of terms in* $\mathcal{T}_{\mathcal{F}}(\mathcal{N})$ *satisfying* $\alpha \to_{\mathcal{G}} \beta$ *if and only if there exists a production* $N \to \gamma$ *in* $\mathcal{P}$ *such that after substituting* γ *for some occurrence of* N *in* α *we obtain* β. *Following standard notational conventions, we use* $\xrightarrow{*}_{\mathcal{G}}$ *to denote the transitive closure of* $\to_{\mathcal{G}}$. *Moreover, if* $\mathcal{G}$ *is clear from the context, we omit it in the subscript of the derivation relations and simply write* $\to$ *and* $\xrightarrow{*}$.

A regular tree grammar $\mathcal{G}$ *with axiom* S *is said to be* unambiguous *if and only if for each ground term* $\alpha \in \mathcal{T}_{\mathcal{F}}(\varnothing)$ *there exists at most one* derivation sequence *in form of* $S \to \gamma_1 \to \cdots \to \gamma_n = \alpha$. *Likewise,* N *is said to be* unambiguous in $\mathcal{G}$ *if and only if for each ground term* $\alpha \in \mathcal{T}_{\mathcal{F}}(\varnothing)$ *there exists at most one derivation sequence in form of* $N \to \gamma_1 \to \cdots \to \gamma_n = \alpha$.

▶ **Definition 2.6** (Regular tree languages). *The* language $L(\mathcal{G})$ generated by $\mathcal{G}$ *is the set of all ground terms α such that $S \xrightarrow{*} \alpha$ where S is the axiom of $\mathcal{G}$. Similarly, the* language generated by term $\alpha \in \mathcal{T}_{\mathcal{F}}(\mathcal{N})$ in $\mathcal{G}$, *denoted as $L_{\mathcal{G}}(\alpha)$, is the set of all ground terms β such that $\alpha \xrightarrow{*} \beta$. Finally, a set $\mathcal{L}$ of ground terms is said to be a* regular tree language *if there exists a regular tree grammar $\mathcal{G}$ such that $L(\mathcal{G}) = \mathcal{L}$.*

▶ **Example 2.7.** The set of $\lambda\upsilon$-terms is an example of a regular tree language. The corresponding regular tree grammar $\Lambda = (T, \mathcal{N}, \mathcal{F}, \mathcal{P})$ consists of

- a set $\mathcal{N}$ of three non-terminal symbols T, S, N intended to stand for $\lambda\upsilon$-terms, substitutions, and de Bruijn indices, respectively, with T being the axiom of Λ;
- a set $\mathcal{F}$ of terminal symbols, comprised of all the symbols of the $\lambda\upsilon$-calculus language, i.e. term application and abstraction, closure $\cdot[\cdot]$, slash $\cdot/$, lift $\Uparrow(\cdot)$ and shift $\uparrow$ operators, and the successor $\mathsf{S}(\cdot)$ with the constant $\underline{0}$; and
- a set $\mathcal{P}$ of productions

$$\begin{aligned} &T \to N \mid \lambda T \mid TT \mid T[S] \\ &S \to T/ \mid \Uparrow(S) \mid \uparrow \\ &N \to \underline{0} \mid \mathsf{S}\,N. \end{aligned} \tag{3}$$

Let us notice that (3) consists of five self-referencing productions, three for T and one for each S and N. Moreover, $L(N) \subset L(T)$ as $\mathcal{P}$ includes a production $T \to N$.

▶ **Example 2.8.** Each $\lambda\upsilon$-term admits a natural tree-like structure. The following example depicts the tree representation of the term $\lambda\underline{1}[\underline{0}[\uparrow][\Uparrow(\underline{0}/)]]$. Note that the (conventionally) implicit term application is represented as an explicit binary node $(\cdot)$.

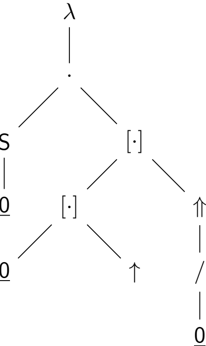


Figure 2 Tree representation of $\lambda\underline{1}[\underline{0}[\uparrow][\Uparrow(\underline{0}/)]]$.

3 Reduction grammars

We conduct our construction of $(\mathcal{G}_n)_n$ in an inductive, incremental fashion. Starting with $\mathcal{G}_0$ corresponding to the set of *pure terms* (i.e. $\lambda\upsilon$-terms without closures) we build the $(n+1)$st grammar $\mathcal{G}_{n+1}$ based on the structure of the nth grammar $\mathcal{G}_n$. First-order rewriting rules of $\lambda\upsilon$-calculus guarantee a close structural resemblance of both their left- and right-hand sides, see Figure 1b. Consequently, with $\mathcal{G}_n$ at hand, we can analyse the right-hand sides of υ rewriting rules and match them with productions of $\mathcal{G}_n$. Based on their structure, we then determine the structure of productions of $\mathcal{G}_{n+1}$ which correspond to respective left-hand sides. Although such a general idea of constructing $\mathcal{G}_{n+1}$ out of $\mathcal{G}_n$ is quite straightforward, its implementation requires some careful amount of detail.

For that reason, we make the following initial preparations. Each grammar $\mathcal{G}_n$ uses the same, global, ranked alphabet $\mathcal{F}$ corresponding to $\lambda\upsilon$-calculus, see Example 2.7. The standard non-terminal symbols T, S, and N, together with their respective productions (3), are *pre-defined* in each grammar $\mathcal{G}_n$. In addition, $\mathcal{G}_n$ includes $n + 1$ non-terminal symbols $G_0, \ldots, G_n$ (the final one being the axiom of $\mathcal{G}_n$) with the intended meaning that for all $0 \leq k \leq n$, the language $L_{\mathcal{G}_n}(G_k)$ is equal to the set of terms υ-normalising in k normal-order steps. In this manner, building $(\mathcal{G}_n)_n$ amounts to a careful, incremental extension process, starting with the initial grammar $\mathcal{G}_0$ comprised of the following extra, i.e. not included in (3), productions:

$$G_0 \to N \mid \lambda G_0 \mid G_0 G_0. \tag{4}$$

In order to formalise the above preparations and the overall presentation of our construction, we introduce the following, abstract notions of υ-reduction grammar and later also their simple variants.

▶ **Definition 3.1** (υ-reduction grammars)**.** *Let $\Lambda = (T, \mathcal{N}, \mathcal{F}, \mathcal{P})$ be the regular tree grammar corresponding to $\lambda\upsilon$-terms, see Example 2.7. Then, the regular tree grammar*

$$\mathcal{G}_n = (G_n, \mathcal{N}_n, \mathcal{F}, \mathcal{P}_n) \tag{5}$$

with

- $\mathcal{N}_n = \mathcal{N} \cup \{G_0, G_1, \ldots, G_n\}$*; and*
- $\mathcal{P}_n$ *being a set of productions such that* $\mathcal{P} \subset \mathcal{P}_n$

is said to be a υ-reduction grammar, υ-RG *in short, if all non-terminal symbols $\mathcal{N}_n$ are unambiguous in $\mathcal{G}_n$, for all $0 \leq k \leq n$ the language $L(G_k)$ is equal to the set of $\lambda\upsilon$-terms υ-normalising in k normal-order υ-steps, and finally $L(T)$, $L(S)$ and $L(N)$ are equal to the sets of $\lambda\upsilon$-terms, substitutions and de Bruijn indices, respectively.*

▶ **Definition 3.2** (Partial order of sorts)**.** *The* partial order of sorts $(\mathcal{N}_n, \preceq)$ *is a partial order (i.e. reflexive, transitive, and anti-symmetric relation) on non-terminal symbols $\mathcal{N}_n$ satisfying $X \preceq Y$ if and only if $L_{\mathcal{G}_n}(X) \subseteq L_{\mathcal{G}_n}(Y)$. For convenience, we write $X \curlywedge Y$ to denote the greatest lower bound of $\{X, Y\}$. Figure 3 depicts the partial order $(\mathcal{N}_n, \preceq)$.*

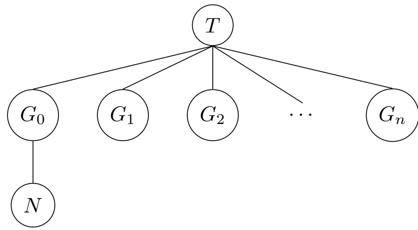


Figure 3 Hasse diagram of the partial order $(\mathcal{N}_n, \preceq)$.

▶ Remark 3.3. Let us notice that given the interpretation of $L(G_0), \ldots, L(G_n)$, the partial order of sorts $(\mathcal{N}_n, \preceq)$ captures all the inclusions among the non-terminal languages within $\mathcal{G}_n$. However, in addition, if X and Y are not comparable through $\preceq$, then $L(X) \cap L(Y) = \varnothing$ as each term υ-normalises in a unique, determined number of steps.

▶ **Definition 3.4** (Simple υ-reduction grammars). *A υ-RG $\mathcal{G}_n$ is said to be* simple *if all its self-referencing productions are either productions of the regular tree grammar corresponding to $\lambda\upsilon$-terms, see Example 2.7, or are of the form*

$$G_k \to \lambda G_k \mid G_0 G_k \mid G_k G_0 \tag{6}$$

and moreover, for all regular productions in form of $G_k \to \alpha$ in $\mathcal{G}_n$, it holds $\alpha \in \mathcal{T}_{\mathcal{F}}(\mathcal{N} \cup \{G_0, \ldots, G_{k-1}\})$, i.e. α does not reference non-terminals other than $G_0, \ldots, G_{k-1}$.

▶ Remark 3.5. Let us note that, in general, υ-reduction grammars do not have to be simple. Due to the erasing (RVar) rewriting rule, it is possible to construct, *inter alia*, more involved self-referencing productions. Nonetheless, for technical convenience, we will maintain the simplicity of constructed grammars $(\mathcal{G}_n)_n$.

Also, let us remark that the above definition of simple υ-reduction grammars asserts that if $\mathcal{G}_{n+1}$ is a simple υ-RG, then, by a suitable truncation, it is possible to obtain all of the υ-reduction grammars $\mathcal{G}_0$ up to $\mathcal{G}_n$. Consequently, $\mathcal{G}_{n+1}$ contains, in a proper sense, all the proceeding grammars $\mathcal{G}_0, \ldots, \mathcal{G}_n$.

4 Finite intersection partitions

The main technical ingredient in our construction of $(\mathcal{G}_n)_n$ are finite intersection partitions.

▶ **Definition 4.1** (Finite intersection partition). *Assume that α, β are two terms in $\mathcal{T}_{\mathcal{F}}(X)$. Then, a* finite intersection partition*,* FIP *in short, of α and β is a finite set $\Pi(\alpha, \beta) = \{\pi_1, \ldots, \pi_n\} \subset \mathcal{T}_{\mathcal{F}}(X)$ such that $L(\pi_i) \cap L(\pi_j) = \varnothing$ whenever $i \neq j$, and moreover $\bigcup_i L(\pi_i) = L(\alpha) \cap L(\beta)$.*

Let us note that, *a priori*, it is not immediately clear if $\Pi(\alpha, \beta)$ exists for α and β in the term algebra $\mathcal{T}(\mathcal{N}_n)$ associated with a simple υ-RG $\mathcal{G}_n$ nor whether there exists an algorithmic method of its construction. The following result states that both questions can be settled in the affirmative.

▶ **Lemma 4.2** (Constructible finite intersection partitions). *Let $\mathcal{G}_n$ be a simple υ-reduction grammar. Assume that α, β are two (not necessarily ground) terms in $\mathcal{T}(\mathcal{N}_n)$ where $\mathcal{N}_n$ is the set of non-terminal symbols of $\mathcal{G}_n$. Then, α and β have a computable finite intersection partition $\Pi(\alpha, \beta)$.*

Figure 4 provides a functional pseudocode description of $\texttt{fip}_k$ constructing $\Pi(\alpha, \beta)$ for arbitrary terms α, β within the scope of a simple υ-RG $\mathcal{G}_k$. A corresponding proof of correctness is given in Appendix A.

Our finite intersection partition algorithm resembles a variant of Robinson's unification algorithm [17] applied to many-sorted term algebras with a tree hierarchy of sorts, as investigated by Walther, cf. [19]. It becomes even more apparent once the correspondence between sorts, as stated in the language of many-sorted term algebra, and the tree-like hierarchy of non-terminal symbols in υ-reduction grammars is established, see Figure 3.

5 The construction of simple υ-reduction grammars

Equipped with constructible, finite intersection partitions, we are now ready to describe the generation procedure for $(\mathcal{G}_n)_n$. We begin with establishing a convenient *verbosity* invariant maintained during the construction of $(\mathcal{G}_n)_n$.

```
1   fun fip_k α β :=
2       match α, β with
3       | f(α_1,...,α_n), g(β_1,...,β_m) ⇒
4           if f ≠ g ∨ n ≠ m then ∅ (* symbol clash *)
5           else if n = m = 0 then {f}
6           else let Π_i := fip_k α_i β_i, ∀ 1 ≤ i ≤ n
7                in {f(π_1,...,π_n) | (π_1,...,π_n) ∈ Π_1 × ··· × Π_n}
8
9       | X, Y ⇒
10          {X ⋏ Y | X ⪯ Y ∨ Y ⪯ X}
11
12      | f(α_1,...,α_n), X ⇒
13          fip_k X f(α_1,...,α_n) (* flip arguments *)
14
15      | X, f(α_1,...,α_n) ⇒
16          let Π_γ := fip_k γ f(α_1,...,α_n), ∀ (X → γ) ∈ 𝒢_k
17              in ⋃_{(X→γ)∈𝒢_k} Π_γ
18  end.
```

Figure 4 Pseudocode of the $\mathtt{fip}_k$ procedure computing $\Pi(\alpha, \beta)$.

▶ **Definition 5.1** (Closure width). *Let α be a term in $\mathcal{T}_{\mathcal{F}}(X)$ for some finite set X. Then, α has* closure width *w, if w is the largest non-negative integer such that α is of form $\chi[\sigma_1]\cdots[\sigma_w]$ for some term χ and substitutions $\sigma_1, \ldots, \sigma_w$. For convenience, we refer to χ as the* head *of α and to $\sigma_1, \ldots, \sigma_w$ as its* tail.

▶ **Definition 5.2** (Verbose υ-reduction grammars). *A υ-RG $\mathcal{G}_n$ is said to be* verbose *if none of its productions is of form $X \to G_k[\sigma_1]\cdots[\sigma_w]$ for some arbitrary non-negative w and k.*

Simple, verbose υ-reduction grammars admit a neat structural feature. Specifically, their productions preserve closure width of generated terms.

▶ **Lemma 5.3.** *Assume that $\mathcal{G}_n$ is a simple, verbose υ-RG. Then, for each production $G_n \to \chi[\sigma_1]\cdots[\sigma_w]$ in $\mathcal{G}_n$ such that its right-hand side is of closure width w, either $\chi = N$ or χ is in form $\chi = f(\alpha_1, \ldots, \alpha_m)$ for some non-closure function symbol f of arity m.*

Proof. See Appendix B. ◀

▶ **Lemma 5.4.** *Assume that $\mathcal{G}_n$ is a simple, verbose υ-RG. Then, for each production $G_n \to \chi[\sigma_1]\cdots[\sigma_w]$ in $\mathcal{G}_n$ such that its right-hand side is of closure width w, and ground term $\delta \in L(\chi[\sigma_1]\cdots[\sigma_w])$ it holds that δ is of closure width w.*

Proof. Direct consequence of Lemma 5.3. ◀

The following φ-matchings are the central tool used in the construction of new reduction grammars. Based on finite intersection partitions, φ-matchings provide a simple *template recognition* mechanism which allows us to match productions in $\mathcal{G}_n$ with right-hand sides of υ rewriting rules.

▶ **Definition 5.5** (φ-matchings)**.** *Let $\mathcal{G}_n$ be a simple, verbose υ-RG and $\varphi = \chi[\tau_1]\cdots[\tau_d] \in \mathcal{T}(\mathcal{N}_n)$ be a* template *(term) of closure width d. Assume that $X \to \gamma[\sigma_1]\cdots[\sigma_w]$ is a* production *of $\mathcal{G}_n$ which right-hand side has closure width $w \geq d$. Furthermore, let*

$$\Delta_\varphi(\gamma[\sigma_1]\cdots[\sigma_w]) = \Pi(\gamma[\sigma_1]\cdots[\sigma_w], \chi[\tau_1]\cdots[\tau_d]\underbrace{[S]\cdots[S]}_{w-d \text{ times}}) \tag{7}$$

*be the set of φ-*matchings *of $\gamma[\sigma_1]\cdots[\sigma_w]$.*

*Then, the set Δ^n_φ of φ-*matchings *of $\mathcal{G}_n$ is defined as*

$$\Delta^n_\varphi = \bigcup\{\Delta_\varphi(\gamma) \mid G_n \to \gamma \in \mathcal{G}_n\}. \tag{8}$$

For further convenience, we write $\varphi_{\langle i\rangle}$ to denote the template $\varphi = \chi[\tau_1]\cdots[\tau_d]$ with i copies of $[S]$ appended to its original tail, i.e. $\varphi_{\langle i\rangle} = \chi[\tau_1]\cdots[\tau_d]\underbrace{[S]\cdots[S]}_{i \text{ times}}$.

Table 1 υ-rewriting rules with respective templates and production schemes.

	Rewriting rule	Template φ	Production scheme $\Delta^n_\varphi \mapsto \gamma$
(App)	$(ab)[s] \to a[s](b[s])$	$T[S](T[S])$	$\alpha[\tau_1](\beta[\tau_2])[\sigma_1]\cdots[\sigma_w] \mapsto (\alpha\beta)[\tau][\sigma_1]\cdots[\sigma_w]^\dagger$
(Lambda)	$(\lambda a)[s] \to \lambda(a[\Uparrow(s)])$	$\lambda(T[\Uparrow(S)])$	$\lambda(\alpha[\Uparrow(\sigma)])[\sigma_1]\cdots[\sigma_w] \mapsto (\lambda\alpha)[\sigma][\sigma_1]\cdots[\sigma_w]$
(FVar)	$\underline{0}[a/] \to a$	T	see Remark 5.8
(RVar)	$(\mathsf{S}\,\underline{n})[a/] \to \underline{n}$	N	$\alpha[\sigma_1]\cdots[\sigma_w] \mapsto (\mathsf{S}\,\alpha)[T/][\sigma_1]\cdots[\sigma_w]$
(FVarLift)	$\underline{0}[\Uparrow(s)] \to \underline{0}$	$\underline{0}$	$\underline{0}[\sigma_1]\cdots[\sigma_w] \mapsto \underline{0}[\Uparrow(S)][\sigma_1]\cdots[\sigma_w]$
(RVarLift)	$(\mathsf{S}\,\underline{n})[\Uparrow(s)] \to \underline{n}[s][\uparrow]$	$N[S][\uparrow]$	$\alpha[\sigma][\uparrow][\sigma_1]\cdots[\sigma_w] \mapsto (\mathsf{S}\,\alpha)[\Uparrow(\sigma)][\sigma_1]\cdots[\sigma_w]$
(VarShift)	$\underline{n}[\uparrow] \to \mathsf{S}\,\underline{n}$	$\mathsf{S}\,N$	$(\mathsf{S}\,\alpha)[\sigma_1]\cdots[\sigma_w] \mapsto \alpha[\uparrow][\sigma_1]\cdots[\sigma_w]$

†For each $\tau \in \Pi(\tau_1, \tau_2)$, see Remark 5.7.

In what follows we use computable intersection partitions in our iterative construction of $(\mathcal{G}_n)_n$. Recall that if $\mathcal{G}_n$ is simple then, *inter alia*, self-referencing productions starting with the non-terminal G_n take the form

$$G_n \to \lambda G_n \mid G_0 G_n \mid G_n G_0. \tag{9}$$

If $t\downarrow_n$ (for $n \geq 1$) but it does not start with a head υ-redex, then it must be of form $t = \lambda a$ or $t = ab$. In the former case, it must hold $a\downarrow_n$; hence the pre-defined production $G_n \to \lambda G_n$ in $\mathcal{G}_n$. In the latter case, it must hold $a\downarrow_k$ whereas $b\downarrow_{n-k}$ for some $0 \leq k \leq n$. And so, it means that we have to include productions in form of $G_n \to G_k G_{n-k}$ for all $0 \leq k \leq n$ in $\mathcal{G}_n$; in particular, the already mentioned two self-referencing productions, see (9).

Remaining terms have to start with head redexes. Each of these head υ-redexes is covered by a dedicated set of productions. The following Lemma 5.6 demonstrates how φ-matchings and, in particular, finite intersection partitions can be used for that purpose.

▶ **Lemma 5.6.** *Let $\varphi = \lambda(T[\Uparrow(S)])$ be the template corresponding to the* (Lambda) *rewriting rule, see Table 1, and $t = (\lambda a)[s][s_1]\cdots[s_w]$. Then, $t\downarrow_{n+1}$ if and only if there exists a unique term $\pi = (\lambda(\alpha[\Uparrow(\sigma)]))[\sigma_1]\cdots[\sigma_w] \in \Delta^n_\varphi$ such that $t \in L((\lambda\alpha)[\sigma][\sigma_1]\cdots[\sigma_w])$.*

Proof. See Appendix B. ◀

Let us remark that almost all of the rewriting rules of $\lambda\upsilon$ exhibit a similar construction scheme; the exceptional (App) and (FVar) rewriting rules are discussed in Remark 5.7 and Remark 5.8, respectively. Given a rewriting rule, we start with the respective template φ (see Table 1) and generate all possible φ-matchings in $\mathcal{G}_n$. Intuitively, such an operation extracts unambiguous sublanguages out of each production in $\mathcal{G}_n$ which match the right-hand side of the considered rewriting rule. Next, we consider each term $\pi \in \Delta_\varphi^n$ and establish new productions $G_{n+1} \to \gamma$ in $\mathcal{G}_{n+1}$ out of π. Assuming that $\mathcal{G}_n$ is a simple and verbose υ-RG, the novel productions generated by means of Δ_φ^n cover all the $\lambda\upsilon$-terms reducing in $n+1$ normal-order steps, starting with the prescribed head rewriting rule. Since the head of each so constructed production either starts with a function symbol or is equal to N, cf. Table 1, the outcome grammar is necessarily verbose. Moreover, if we complete the production generation for all rewriting rules, by construction, the grammar $\mathcal{G}_{n+1}$ must be, at the same time, simple. Consequently, the construction of the hierarchy $(\mathcal{G}_n)_n$ amounts to an inductive application of the above construction scheme.

▶ Remark 5.7. While following the same pattern for the (App) rule, we notice that the corresponding construction requires a slight modification. Specifically, while matching $\varphi = T[S](T[S])$ with a right-hand side γ of a production $G_n \to \gamma$ in $\mathcal{G}_n$ we cannot conclude that $\pi \in \Delta_\varphi^n$ takes the form $\pi = \alpha[\sigma](\beta[\sigma])[\sigma_1]\cdots[\sigma_w]$. Note that, in fact, we know that $\pi = \alpha[\tau_1](\beta[\tau_2])[\sigma_1]\cdots[\sigma_w]$ however perhaps $\tau_1 \neq \tau_2$. Nonetheless, we can still compute $\Pi(\tau_1,\tau_2)$ and use $\tau \in \Pi(\tau_1,\tau_2)$ to generate a finite set of terms in form of $\pi = \alpha[\tau](\beta[\tau])[\sigma_1]\cdots[\sigma_w]$. Using those terms, we can continue with our construction and establish a set of new productions in form of $G_n \to (\alpha\beta)[\tau][\sigma_1]\cdots[\sigma_w]$ meant to be included in $\mathcal{G}_{n+1}$.

▶ Remark 5.8. Let us also remark that the single rewriting rule which has a template φ not retaining closure width is (FVar). In consequence, the utility of $\Delta_T(\gamma)$ is substantially limited. If $t = \underline{0}[a/][s_1]\cdots[s_w] \downarrow_{n+1}$, then $t \to t' = a[s_1]\cdots[s_w]$ which, in turn, satisfies $t' \downarrow_n$. Note that if γ is the right-hand side of a unique production $G_n \to \gamma$ in $\mathcal{G}_n$ generating t', then we can match T with *any* non-empty prefix of γ. The length of the chosen prefix influences what initial part α of γ is going to be placed under the closure in $G_{n+1} \to \underline{0}[\alpha/][\sigma_1]\cdots[\sigma_w]$.

This motivates the following approach. Let $G_n \to \gamma' = \chi[\sigma_1]\cdots[\sigma_w]$ be an arbitrary production in $\mathcal{G}_n$ of closure width w. If $t' \in L(\gamma')$ and $t \to t'$ in a single head (FVar)-reduction, then $t \in L(\underline{0}[\chi[\sigma_1]\cdots[\sigma_d]/][\sigma_{d+1}]\cdots[\sigma_w]$ for some $0 \le d \le w$. Therefore, in order to generate all productions in $\mathcal{G}_{n+1}$ corresponding to $\lambda\upsilon$-terms υ-normalising in $n+1$ steps, starting with a head (FVar)-reduction, we have to include all productions in form of $G_{n+1} \to \underline{0}[\chi[\sigma_1]\cdots[\sigma_d]/][\sigma_{d+1}]\cdots[\sigma_w]$ for each production $G_n \to \gamma'$ in $\mathcal{G}_n$.

Finally, note that it is, again, possible to optimise the (FVar) construction scheme with respect to the number of generated productions. For each $G_n \to \gamma$ in $\mathcal{G}_n$ the above scheme produces, *inter alia*, a production $G_{n+1} \to \underline{0}[\gamma/]$. Note that we can easily merge them into a single production $G_{n+1} \to \underline{0}[G_n/]$ instead.

Such a construction leads us to the following conclusion.

▶ **Theorem 5.9.** *For all $n \ge 0$ there exists a constructible, simple υ-RG $\mathcal{G}_n$.*

▶ **Example 5.10.** The following example demonstrates the construction of $\mathcal{G}_1$ out of $\mathcal{G}_0$. Note that $\mathcal{G}_1$ includes the following productions associated with the axiom G_1:

$$\begin{aligned} G_1 \to\ & \lambda G_1 \mid G_0 G_1 \mid G_1 G_0 \\ & \mid \underline{0}[(G_0 G_0)/] \mid \underline{0}[\lambda G_0/] \mid \underline{0}[N/] \\ & \mid (\mathsf{S}\, N)[T/] \mid \underline{0}[\Uparrow (S)] \mid N[\uparrow]. \end{aligned} \tag{10}$$

The first three productions are included by default. The next three productions are derived from the (FVar) rule applied to all the productions of $G_0 \to \gamma$ in $\mathcal{G}_0$. The final three productions are obtained by (RVar), (FVarLift), and (VarShift), respectively.

6 Analytic combinatorics and simple υ-reduction grammars

Having established an effective hierarchy $(\mathcal{G}_k)_k$ of simple υ-reduction grammars, we can now proceed with their quantitative analysis. Given the fact that regular tree grammars represent well-known algebraic tree-like structures, our analysis is in fact a standard application of algebraic singularity analysis of respective generating functions [12, 13]. The following result provides the main tool of the current section.

▶ **Proposition 6.1** (Algebraic singularity analysis, see [13], Theorem VII.8)**.** *Assume that* $f(z) = \left(\sqrt{1 - z/\zeta}\right) g(z) + h(z)$ *is an algebraic function, analytic at* 0*, and has a unique dominant singularity* $z = \zeta$*. Moreover, assume that* $g(z)$ *and* $h(z)$ *are analytic in the disk* $|z| < \zeta + \varepsilon$ *for some* $\varepsilon > 0$*. Then, the coefficients* $[z^n]f(z)$ *in the series expansion of* $f(z)$ *around the origin, satisfy the following asymptotic estimate*

$$[z^n]f(z) \sim \zeta^{-n} \frac{n^{-3/2} g(\zeta)}{\Gamma(-\frac{1}{2})}. \tag{11}$$

In order to analyse the number of $\lambda\upsilon$-terms normalising in k steps, we execute the following plan. First, we use the structure (and unambiguity) of $\mathcal{G}_k$ to convert it by means of symbolic methods into a corresponding generating function $G_k(z) = \sum g_n^{(k)} z^n$ in which the integer coefficient $g_n^{(k)}$ standing by z^n in the series expansion of $G_k(z)$, also denoted as $[z^n]G_k(z)$, is equal to the number of $\lambda\upsilon$-terms of size n normalising in k steps. Next, we show that so obtained generating functions fit the premises of Proposition 6.1. We start with establishing an appropriate size notion for $\lambda\upsilon$-terms. For technical convenience, we assume the following *natural size notion*, equivalent to the number of constructors in the associated term algebra $\mathcal{T}_{\mathcal{F}}(\varnothing)$, see Figure 5.

$$\begin{aligned} |\underline{\mathsf{n}}| &= n + 1 \\ |\lambda a| &= 1 + |a| \\ |ab| &= 1 + |a| + |b| \\ |a[s]| &= 1 + |a| + |s| \end{aligned} \qquad\qquad \begin{aligned} |a/| &= 1 + |a| \\ |\Uparrow(s)| &= 1 + |s| \\ |\uparrow| &= 1. \end{aligned}$$

Figure 5 Natural size notion for $\lambda\upsilon$-terms, cf. Figure 2.

The following results exhibit the closed-form of generating functions corresponding to pure terms as well as the general class of $\lambda\upsilon$-terms and explicit substitutions.

▶ **Proposition 6.2** (see [5])**.** *Let* $L_\infty(z)$ *denote the generating function corresponding to the set of* λ*-terms in* υ*-normal form (i.e. without* υ*-redexes). Then,*

$$L_\infty(z) = \frac{1 - z - \sqrt{\frac{1-3z-z^2-z^3}{1-z}}}{2z}. \tag{12}$$

▶ **Proposition 6.3** (see [7])**.** *Let* $T(z)$*,* $S(z)$ *and* $N(z)$ *denote the generating functions corresponding to* $\lambda\upsilon$*-terms, substitutions, and de Bruijn indices, respectively. Then,*

$$T(z) = \frac{1 - \sqrt{1-4z}}{2z} - 1, \quad S(z) = \frac{1 - \sqrt{1-4z}}{2z} \left(\frac{z}{1-z}\right) \quad \textit{and} \quad N(z) = \frac{z}{1-z}. \tag{13}$$

With the above basic generating functions, we can now proceed with the construction of generating functions corresponding to simple υ-reduction grammars. See Appendix C for respective proofs.

▶ **Proposition 6.4** (Constructible generating functions). *Let Φ_k denote the set of regular productions in $\mathcal{G}_k$. Then, for all $k \geq 1$ there exists a generating function $G_k(z)$ such that $[z^n]G_k(z)$ (i.e. the coefficient standing by z^n in the power series expansion of $G_k(z)$) is equal to the number of terms of size n which υ-normalise in k normal-order reduction steps, and moreover $G_k(z)$ admits a closed-form of the following shape:*

$$G_k(z) = \frac{1}{1 - z - 2zL_\infty(z)} \sum_{G_k \to \gamma \in \Phi_k} G_\gamma(z) \tag{14}$$

where

$$G_\gamma(z) = z^{\zeta(\gamma)} T(z)^{\tau(\gamma)} S(z)^{\sigma(\gamma)} N(z)^{\nu(\gamma)} \prod_{0 \leq i < k} G_i(z)^{\rho_i(\gamma)} \tag{15}$$

and all $\zeta(\gamma)$, $\tau(\gamma)$, $\sigma(\gamma)$, $\nu(\gamma)$, and $\rho_i(\gamma)$ are non-negative integers depending on γ.

▶ **Theorem 6.5.** *For all $k \geq 1$, the coefficients $[z^n]G_k(z)$ satisfy the asymptotic estimate*

$$[z^n]G_k(z) \sim c_k \cdot 4^n n^{-3/2}. \tag{16}$$

Consider the following *asymptotic density* of $\lambda\upsilon$-terms υ-normalisable in k normal-order reduction steps in the set of all $\lambda\upsilon$-terms:

$$\mu_k = \lim_{n \to \infty} \frac{[z^n]G_k(z)}{[z^n]T(z)}. \tag{17}$$

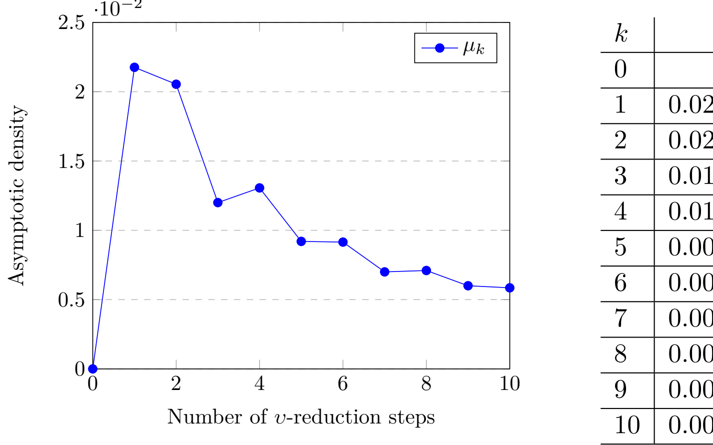
In other words, the limit μ_k of the probability that a uniformly random $\lambda\upsilon$-term of size n normalises in k steps as n tends to infinity. Note that for each $k \geq 1$, the asymptotic density μ_k is positive as both $[z^n]G_k(z)$ and $[z^n]T(z)$ admit the same (up to a multiplicative constant) asymptotic estimate. Moreover, it holds $\mu_k \xrightarrow[k\to\infty]{} 0$ as the sum $\sum_k \mu_k$ is increasing and necessarily bounded above by one.

Figure 6 provides the specific asymptotic densities $\mu_0, \ldots, \mu_{10}$ obtained by means of a direct construction and symbolic computations[1] (numerical values are rounded up to the fifth decimal point).

▶ Remark 6.6. Theorem 5.9 and Theorem 6.5 are effective in the sense that both the symbolic representation and the symbolic asymptotic estimate of respective coefficients are computable. Since $\Gamma(-1/2) = -2\sqrt{\pi}$ is the sole transcendental number occurring in the asymptotic estimates, and cancels out when asymptotic densities are considered, cf. (17), we immediately note that for each $k \geq 0$, the asymptotic density of terms υ-normalising in k steps is necessarily an algebraic number.

▶ Remark 6.7. Each $\lambda\upsilon$-term is υ-normalising in some (determined) number of normal-order reduction steps. However, it is not clear whether $\sum_k \mu_k = 1$ as asymptotic density is, in general, not countably additive. Let us remark that if this sum converges to one, then the random variable X_n denoting the number of normal-order υ-reduction steps required to normalise a random $\lambda\upsilon$-term of size n (i.e. the average-case cost of resolving pending substitutions in a random term of size n) converges pointwise to a discrete random variable X defined as $\mathbb{P}(X = k) = \mu_k$. Alas, our current analysis does not determine an answer to this problem.

[1] Corresponding software is available at `https://github.com/maciej-bendkowski/towards-acasrlc`.



k	μ_k
0	0.
1	0.02176
2	0.02054
3	0.01200
4	0.01306
5	0.00920
6	0.00915
7	0.00700
8	0.00710
9	0.00600
10	0.00585

Figure 6 Asymptotic densities of terms υ-normalising in k normal-order reduction steps. In particular, we have $\mu_0 + \cdots + \mu_{10} \approx 0.11162$.

7 Applications to other term rewriting systems

Let us note that the presented construction of reduction grammars does not depend on specific traits immanent to $\lambda\upsilon$, but rather on certain more general features of its rewriting rules. The key ingredient in our approach is the ability to compute finite intersection partitions for arbitrary terms within the scope of established reduction grammars, which themselves admit certain neat structural properties. Using finite intersection partitions, it becomes possible to generate new productions based on the structural resemblance of both the left-hand and right-hand sides of associated rewriting rules. In what follows we sketch the application of the presented approach to other term rewriting systems, focusing on two examples, i.e. $\lambda\upsilon$-calculus and combinatory logic. Although the technique is similar, some careful amount of details is required.

7.1 $\lambda\upsilon$-calculus

In order to characterise the full $\lambda\upsilon$-calculus, we need a few adjustments to the already established construction of $(\mathcal{G}_k)_k$ corresponding to the υ fragment. Clearly, we have to establish a new production construction scheme associated with the (Beta) rewriting rule. Consider the corresponding template $\varphi = T[T/]$. Like the respective template for (FVar), cf. Remark 5.8, the current template φ does not retain closure width of generated ground terms. However, at the same time it is also amenable to similar treatment as (FVar).

Let $t = (\lambda a)b[s_1]\cdots[s_w] \downarrow_{n+1}$. Note that $t \to a[b/][s_1]\cdots[s_w] \downarrow_n$. Consequently, one should attempt to match φ with all possible *prefixes* of γ in all productions $G_n \to \gamma$ instead of merely their heads, as in the case of Δ_φ^n. Let $\gamma = \chi[\sigma_1]\cdots[\sigma_d]$ be of closure width d and $\delta = \chi[\sigma_1]\cdots[\sigma_i]$ be its prefix $(1 \leq i \leq d)$. Note that, effectively, $\Pi(\delta, T[T/])$ checks if $[\sigma_i]$ takes form $\beta/$ for some term β. Hence, for each $\pi \in \Pi(\delta, T[T/])$ we have $\pi = \alpha[\tau_1]\cdots[\tau_{i-1}][\beta/]$ for some terms α, $\tau_1, \ldots, \tau_{i-1}$, and β. Out of such a partition π we can then construct the production $G_{n+1} \to (\lambda\alpha[\tau_1]\cdots[\tau_{i-1}])\beta[\tau_{i+1}]\cdots[\tau_d]$ in $\mathcal{G}_{n+1}$.

However, with the new scheme for (Beta) we are not yet done with our construction. Note that due to the new head redex type, we are, *inter alia*, forced to change the pre-defined set of productions corresponding to $\lambda\upsilon$-terms without a head redex. Indeed, note that if t does not start with a head redex, then it must be of form (λa) or (ab) where in the latter case

a cannot start with a leading abstraction. This constraint suggests the following approach. We split the set of productions in form of $G_n \to \gamma$ into two categories, i.e. productions whose right-hand sides are of form $\lambda\gamma'$ and the remainder ones. Since both sets are disjoint, we can group them separately and introduce two auxiliary non-terminal symbols $G_n^{(\lambda)}$ and $G_n^{(\neg\lambda)}$ for terms starting with a head abstraction and for those without a head abstraction, respectively, with the additional productions $G_n \to G_n^{(\lambda)} \mid G_n^{(\neg\lambda)}$. In doing so, it is possible to pre-define the all the productions corresponding to terms without head redexes using productions in form of

$$G_n \to \lambda G_n \mid G_k^{(\neg\lambda)}\ G_{n-k} \qquad \text{where} \qquad 0 \leq k \leq n. \tag{18}$$

This operation, however, requires a minor adjustment in the formal argument concerning the termination and correctness of constructed finite intersection partitions, see Appendix A and, in particular, Definition A.1). Afterwards, it becomes possible to reuse already established techniques in the construction of new productions in $\mathcal{G}_{n+1}$ out of $\mathcal{G}_n$.

7.2 SK-combinators

Using a similar approach, it becomes possible to construct appropriate reduction grammars for SK-combinators. In particular, our current technique (partially) subsumes, and also simplifies, the results of [6, 4]. With merely two rewriting rules in form of $Kxy \to x$ and $Sxyz \to xz(yz)$ we can use the developed finite intersection partitions and φ-matchings to construct a hierarchy $(\mathcal{G}_n)_n$ of normal-order reduction grammars for SK-combinators. The rewriting rule corresponding to K is similar to (FVar) whereas the respective rule for S resembles the (App) rule; as in this case, we have to deal with variable duplication on the right-hand side of the rewriting rule. Instead of closure width, we use a different normal form of terms, and so also productions, based on the sole binary constructor of term application. Consequently, a combinator is of *application width* w if it takes the form $X\alpha_1 \ldots \alpha_w$ for some primitive combinator $X \in \{S, K\}$. Consider the more involved case of productions corresponding to head S-redexes. Let $t = Sxyz\alpha_1 \ldots \alpha_w$ be a term of application width $w + 3$ where $w \geq 0$. Note that

$$Sxyz\delta_1 \ldots \delta_w \to xz(yz)\delta_1 \ldots \delta_w. \tag{19}$$

Let us rewrite the right-hand side of (19) as $t' = Xx_1 \ldots x_k z(yz)\delta_1 \ldots \delta_w$ where $x = Xx_1 \ldots x_k$ and X is a primitive combinator. Assume that γ is the right-hand side of the unique production $G_n \to \gamma$ in $\mathcal{G}_n$ such that $t' \in L(\gamma)$. Note that the shape of t' suggests a construction scheme similar to the already discussed (App), see Remark 5.7, where we first have to match the pattern $\varphi = T(TT)$ with some arguments of γ and subsequently attempt to extract a finite intersection partition $\Pi(\alpha, \beta)$ of respective subterms α and β so that for each $\pi \in \Pi(\alpha, \beta)$ we have $z \in L(\pi)$. With appropriate terms at hand, we can then construct corresponding productions in the next grammar $\mathcal{G}_{n+1}$.

8 Conclusions

Quantitative aspects of term rewriting systems are not well studied. A general complexity analysis was undertaken by Choppy, Kaplan, and Soria who considered a class of confluent, terminating term rewriting systems in which the evaluation cost, measured in the number of rewriting steps required to reach the normal form, is independent of the assumed evaluation strategy [8]. More recently, typical evaluation cost of normal-order reduction in combinatory

logic was studied by Bendkowski, Grygiel and Zaionc [6, 4]. Using quite different, non-analytic methods, Sin'Ya, Asada, Kobayashi and Tsukada considered certain asymptotic aspects of β-reduction in the simply-typed variant of λ-calculus showing that, typically, λ-terms of order k have $(k-1)$-fold exponentially long β-reduction sequences [18].

Arguably, the main novelty in the presented approach lies in the algorithmic construction of reduction grammars $(\mathcal{G}_k)_k$ based on finite intersection partitions, assembled using a general technique reminiscent of Robinson's unification algorithm applied to many-sorted term algebras, cf. [17, 19]. Equipped with finite intersection partitions, the construction of $\mathcal{G}_{k+1}$ out of $\mathcal{G}_k$ follows a stepwise approach, in which new productions are established on a per rewriting rule basis. Consequently, the general technique of generating reduction grammars does not depend on specific features of $\lambda\upsilon$, but rather on more general traits of certain first-order rewriting systems. Nonetheless, the full scope of our technique is yet to be determined.

Although the presented construction is based on the leftmost-outermost reduction scheme, it does not depend on the specific size notion associated with $\lambda\upsilon$-terms; in principle, more involved size models can be assumed and analysed. The assumed evaluation strategy, size notion, as well as the specific choice of $\lambda\upsilon$ are clearly arbitrary and other, equally perfect choices for modelling substitution resolution could have been made. However, due to merely eight rewriting rules forming $\lambda\upsilon$, it is one of the conceptually simplest calculus of explicit substitutions. Together with the normal-order evaluation tactic, it is therefore one of the simplest to investigate in quantitative terms and to demonstrate the finite intersection partitions technique.

Due to the unambiguity of constructed grammars $(\mathcal{G}_k)_k$ it is possible to automatically establish their corresponding combinatorial specifications and, in consequence, obtain respective generating functions encoding sequences $\left(g_n^{(k)}\right)_n$ comprised of numbers $g_n^{(k)}$ associated with $\lambda\upsilon$-terms of size n which reduce in k normal-order rewriting steps to their υ-normal forms. Singularity analysis provides then the means for systematic, quantitative investigations into the properties of substitution resolution in $\lambda\upsilon$, as well as its machine-independent operational complexity. Finally, with generating functions at hand, it is possible to undertake a more sophisticated statistical analysis of substitution (in particular υ-normalisation) using available techniques of analytic combinatorics, effectively analysing the average-case cost of λ-calculus and related term rewriting systems.

It should be noted that such an analysis might provide some novel insight into the combinatorial structure of substitution and, in the long-term perspective, provide a theoretical model for the operational, average-case analysis of substitution in modern functional programming languages. In view of these objectives, we conclude the paper with the following conjecture, postulating the existence of a limit distribution associated with the average-case cost of substitution resolution, and its close relation with the constructible hierarchy of reduction grammars and established series of asymptotic densities.

▶ **Conjecture.** *Let μ_k denote the asymptotic density of $\lambda\upsilon$-terms υ-normalising in k leftmost-outermost υ-reduction steps, cf. Remark 6.7. Then, it holds*

$$\sum_{k \geq 0} \mu_k = 1. \tag{20}$$

Consequently, the sequence $(X_n)_n$ of random variables corresponding to the number of υ-reductions required to normalise a uniformly random $\lambda\upsilon$-term of size n (i.e. the average-case cost of resolving all pending substitutions in a random $\lambda\upsilon$-term of size n) converges pointwise to a random variable X satisfying $\mathbb{P}(X = k) := \mu_k$.

References

1. M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991. doi:10.1017/S0956796800000186.
2. B. Accattoli and U. Dal Lago. (Leftmost-Outermost) Beta Reduction is Invariant, Indeed. *Logical Methods in Computer Science*, 12(1), 2016. doi:10.2168/LMCS-12(1:4)2016.
3. Z.-E.-A. Benaissa, D. Briaud, P. Lescanne, and J. Rouyer-Degli. $\lambda\upsilon$, a calculus of explicit substitutions which preserves strong normalisation. *Journal of Functional Programming*, 6(5):699–722, 1996. doi:10.1017/S0956796800001945.
4. M. Bendkowski. Normal-order reduction grammars. *Journal of Functional Programming*, 27, 2017. doi:10.1017/S0956796816000332.
5. M. Bendkowski, K. Grygiel, P. Lescanne, and M. Zaionc. Combinatorics of λ-terms: a natural approach. *Journal of Logic and Computation*, 27(8):2611–2630, 2017. doi:10.1093/logcom/exx018.
6. M. Bendkowski, K. Grygiel, and M. Zaionc. On the likelihood of normalization in combinatory logic. *Journal of Logic and Computation*, 2017. doi:10.1093/logcom/exx005.
7. M. Bendkowski and P. Lescanne. Combinatorics of Explicit Substitutions. In *Proceedings of the 20th International Symposium on Principles and Practice of Declarative Programming*, PPDP '18, pages 7:1–7:12. ACM, 2018. doi:10.1145/3236950.3236951.
8. C. Choppy, S. Kaplan, and M. Soria. Complexity Analysis of Term-Rewriting Systems. *Theoretical Computer Science*, 67(2&3):261–282, 1989. doi:10.1016/0304-3975(89)90005-4.
9. A. Cichon and P. Lescanne. Polynomial interpretations and the complexity of algorithms. In *Automated Deduction—CADE-11*, pages 139–147. Springer Berlin Heidelberg, 1992.
10. H. Comon, M. Dauchet, R. Gilleron, Ch. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree Automata Techniques and Applications, 2007. Release October, 12th 2007. URL: http://www.grappa.univ-lille3.fr/tata.
11. N. G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae (Proceedings)*, 75(5):381–392, 1972.
12. Ph. Flajolet and A. M. Odlyzko. Singularity Analysis of Generating Functions. *SIAM Journal on Discrete Mathematics*, 3(2):216–240, 1990.
13. Ph. Flajolet and R. Sedgewick. *Analytic Combinatorics.* Cambridge University Press, 1 edition, 2009.
14. J. L. Lawall and H. G. Mairson. Optimality and Inefficiency: What Isn't a Cost Model of the Lambda Calculus? In *Proceedings of the 1st ACM SIGPLAN International Conference on Functional Programming*, ICFP '96, pages 92–101. ACM, 1996. doi:10.1145/232627.232639.
15. P. Lescanne. From $\lambda\sigma$ to $\lambda\upsilon$: A Journey Through Calculi of Explicit Substitutions. In *Proceedings of the 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 60–69. ACM, 1994.
16. P. Lescanne. The lambda calculus as an abstract data type. In M. Haveraaen, O. Owe, and O.-J. Dahl, editors, *Recent Trends in Data Type Specification*, pages 74–80. Springer Berlin Heidelberg, 1996.
17. J. A. Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *J. ACM*, 12(1):23–41, January 1965. doi:10.1145/321250.321253.
18. R. Sin'Ya, K. Asada, N. Kobayashi, and T. Tsukada. Almost Every Simply Typed λ-Term Has a Long β-Reduction Sequence. In *Proceedings of the 20th International Conference on Foundations of Software Science and Computation Structures*, volume 10203, pages 53–68. Springer-Verlag New York, Inc., 2017. doi:10.1007/978-3-662-54458-7_4.
19. Ch. Walther. Many-sorted Unification. *J. ACM*, 35(1):1–17, January 1988. doi:10.1145/42267.45071.

A Correctness of the finite intersection partition algorithm

In order to prove the correctness of $\texttt{fip}_k$ constructing $\Pi(\alpha,\beta)$ for arbitrary terms α,β within the scope of a simple υ-RG $\mathcal{G}_k$, see Lemma 4.2, we resort to the following technical notion of *term potential* used to embed terms into the well-founded set of natural numbers.

▶ **Definition A.1** (Term potential)**.** *Let $\alpha \in \mathcal{T}(\mathcal{N}_n)$ be a term and $\mathcal{G}_n$ be a simple υ-RG. Let $\mathsf{Prod}_{\mathcal{G}_n}(X)$ denote the set consisting of right-hand sides of regular productions in form of $X \to \beta$ in $\mathcal{G}_n$. Then, the* potential $\pi(\alpha)$ *of α in $\mathcal{G}_n$ is defined inductively as follows:*

- *If $\alpha = f(\alpha_1,\ldots,\alpha_m)$, then $\pi(\alpha) = 1 + \sum_{i=1}^{m} \pi(\alpha_i)$;*
- *If $\alpha = X \in \{T,S,N\}$, then $\pi(\alpha) = 1 + \max\{\pi(\gamma) \mid \gamma \in \mathsf{Prod}_{\mathcal{G}_n}(X)\}$;*
- *If $\alpha = G_k$ for some $0 \le k \le n$, then $\pi(\alpha) = 1 + \max\{\pi(\gamma) \mid \gamma \in \bigcup_{i=0}^{k} \mathsf{Prod}_{\mathcal{G}_n}(G_i)\}$.*

Let us note that π is well-defined as, by assumption, $\mathsf{Prod}_{\mathcal{G}_n}(X) \neq \varnothing$ for all simple υ-reduction grammars $\mathcal{G}_n$; otherwise $L(G_k)$ could not span the whole set of $\lambda\upsilon$-terms υ-normalising in k normal-order steps. Moreover, π has the following crucial properties:

- *For each term α we have $\pi(\alpha) \ge 1$;*
- *If α is a proper subterm of β, then $\pi(\alpha) < \pi(\beta)$;*
- *If $X \to \alpha$ is a regular production, then $\pi(\alpha) < \pi(X)$; and*
- *For each G_i and G_j, it holds $\pi(G_i) \le \pi(G_j)$ whenever $i \le j$.*

▶ **Example A.2.** Note that the term potential of N associated with de Bruijn indices is equal to $\pi(N) = 2$ as $\pi(\underline{0}) = 1$. Since $T \to N$ is the single regular production starting with T on its left-hand side, the potential $\pi(T)$ is therefore equal to 3. Consequently, we also have $\pi(S) = 5$ as witnessed by the regular production $S \to T/$. Finally, since $\pi(N) = 2$ it holds $\pi(G_0) = 3$ and so, for instance, we also have $\pi(G_0 G_0) = 7$.

Productions of a simple $\mathcal{G}_n$ cannot reference non-terminals other than $G_0,\ldots,G_n$. Since the potential of G_{k+1} is defined in terms of the potential of its regular productions, this means that $\pi(G_{k+1})$ depends, in an implicit manner, on the potentials $\pi(G_0),\ldots,\pi(G_k)$. Note that this constitutes a traditional inductive definition. In order to compute the potential of a given term α, we start with computing the potential of associated non-terminals. In particular, we find the values $\pi(G_0),\ldots,\pi(G_n)$ in ascending order. Afterwards, we can recursively decompose α and calculate its potential based on the potential of non-terminal symbols occurring in α. Note that the same scheme holds, in particular, for the right-hand sides of self-referencing productions.

▶ **Definition A.3** (Conservative productions)**.** *A self-referencing production $X \to f(\alpha_1,\ldots,\alpha_n)$ is said to be* conservative *if $\pi(\alpha_i) \le \pi(X)$ for all $1 \le i \le n$.*

▶ Remark A.4. Conservative productions play a central role in the algorithmic construction of finite intersection partitions $\Pi(\alpha,\beta)$. In particular, let us remark that all self-referencing productions of a simple υ-RG $\mathcal{G}_n$, as listed in Figure 7, are at the same time conservative.

With the technical notions of term potential and conservative productions, we are now ready to prove the correctness of $\texttt{fip}_k$, see Figure 4.

Proof of Lemma 4.2. Induction over the total potential of α and β.

Let us start with the base case $\pi(\alpha) + \pi(\beta) = 2$. Note that both α and β have to be constant ground terms (the potential of non-terminals in $\mathcal{N}_k$ is at least 2). If $\alpha \neq \beta$, then certainly $L(\alpha) \cap L(\beta) = \varnothing$ and so $\Pi(\alpha,\beta) = \varnothing$, see Line 4. Otherwise if $\alpha = \beta$, then both

$$G_k \to \lambda G_k \mid G_0 G_k \mid G_k G_0 \qquad S \to \Uparrow(S)$$
$$T \to \lambda T \mid TT \mid T[S] \qquad N \to \mathsf{S}\, N$$

Figure 7 Self-referencing productions of simple υ-reduction grammars.

$L(\alpha) \cap L(\beta) = \Pi(\alpha, \beta) = \{\alpha\} = \{\beta\}$; hence, $\mathtt{fip}_k$ returns a correct intersection partition $\Pi(\alpha, \beta)$, see Line 5. And so, assume that $\pi(\alpha) + \pi(\beta) > 2$. Depending on the joint structure of both α and β we have to consider three cases.

Case 1. Suppose that $\alpha = f(\alpha_1, \ldots, \alpha_n)$ and $\beta = g(\beta_1, \ldots, \beta_m)$ for some function symbols f and g with $n, m \geq 0$. Certainly, if either $f \neq g$ or $n \neq m$, then $L(\alpha) \cap L(\beta) = \varnothing$. In consequence, $\varnothing$ is the sole valid FIP of both α and β, see Line 4.
So, let us assume that both $f = g$ and $n = m$. Moreover, we can also assume that $n, m \geq 1$ as the trivial case $n = m = 0$ cannot occur under the working assumption $\pi(\alpha) + \pi(\beta) > 2$. Take an arbitrary $\delta \in L(\alpha) \cap L(\beta)$. Note that δ takes the form of $\delta = f(\delta_1, \ldots, \delta_n)$ for some ground terms $\delta_1, \ldots, \delta_n$. By induction, for all $1 \leq i \leq n$, the recursive call $\mathtt{fip}_k\, \alpha_i\, \beta_i$ yields a finite intersection partition $\Pi(\alpha_i, \beta_i)$ of α_i and β_i. Since $\delta_i \in L(\alpha_i) \cap L(\beta_i)$ there exists a unique $\pi_i \in \Pi(\alpha_i, \beta_i)$ such that $\delta_i \in L(\pi_i)$. Accordingly, for $\delta = f(\delta_1, \ldots, \delta_n)$ there exists a unique term $\pi = f(\pi_1, \ldots, \pi_n)$ in $\mathtt{fip}_k\, \alpha\, \beta$ such that $\delta \in L(\pi)$.
Conversely, take an arbitrary ground term $\delta = f(\delta_1, \ldots, \delta_n) \in L(\pi)$ for some $\pi \in \mathtt{fip}_k\, \alpha\, \beta$. Note that π takes the form $\pi = f(\pi_1, \ldots, \pi_n)$, see Line 7. Since $\delta \in L(\pi)$, we know that $\delta_i \in L(\pi_i)$, for each $1 \leq i \leq n$. Moreover, $\pi_i \in \Pi_i$ by the construction of $\mathtt{fip}_k\, \alpha\, \beta$, see Line 6. Following the inductive hypothesis that Π_i is a FIP of α_i and β_i, we notice that $L(\pi_i) \subseteq L(\alpha_i) \cap L(\beta_i)$. Consequently, $\delta_i \in L(\alpha_i) \cap L(\beta_i)$ and so $\delta \in L(\alpha) \cap L(\beta)$.

Case 2. Suppose that $\alpha = X$ and $\beta = Y$ are two non-terminal symbols in $\mathcal{N}_k$, see Line 9. Let us consider the sort poset $(\mathcal{N}_k, \preceq)$ associated with $\mathcal{G}_k$. Assume that X and Y are comparable through $\preceq$ (w.l.o.g. let $X \preceq Y$). Consequently, $L(X) \subseteq L(Y)$ and so $X \curlywedge Y = X$. Clearly, $\{X\}$ is a valid FIP, see Line 10. On the other hand, if X and Y are incomparable in the sort poset associated with $\mathcal{G}_k$, it means that $L(X) \cap L(Y) = \varnothing$ and so $\Pi(\alpha, \beta) = \varnothing$, see Remark 3.3.

Case 3. Suppose w.l.o.g that $\alpha = X$ and β takes the form $\beta = f(\beta_1, \ldots, \beta_n)$ with $n \geq 0$. Note that $\mathtt{fip}_k$ flips its arguments if necessary, see Line 13. Take an arbitrary $\delta \in L(\alpha) \cap L(\beta)$. Note that from the form of β we know that $\delta = f(\delta_1, \ldots, \delta_n)$ for some ground terms $\delta_1, \ldots, \delta_n$ ($n \geq 0$). Since $\alpha = X$ is a non-terminal symbol which, by assumption, is unambiguous in $\mathcal{G}_k$, there exists a unique production $X \to \gamma$ such that $\delta \in L(\gamma)$.
If $X \to \gamma$ is regular (i.e. X does not occur in γ), then $\pi(\gamma) < \pi(X)$ and so $\pi(\gamma) + \pi(\beta) < \pi(X) + \pi(\beta)$. Hence, by induction, $\mathtt{fip}_k\, \gamma\, \beta$ constructs a finite intersection partition $\Pi(\gamma, \beta)$ with a unique $\pi \in \Pi(\gamma, \beta) \subset \Pi(X, \beta)$ such that $\delta \in L(\pi)$, see Line 17.
Let us therefore assume that $X \to \gamma$ is not regular, but instead self-referencing (i.e. X occurs in γ). In such a case $\pi(X) \leq \pi(\gamma)$ and so we cannot directly apply the induction hypothesis to $\mathtt{fip}_k\, \gamma\, \beta$. Note however, that since $\delta \in L(\gamma) \cap L(\beta)$ and $\gamma \neq X$, the term γ must be of form $\gamma = f(\gamma_1, \ldots, \gamma_n)$ as otherwise $\delta \notin L(\gamma)$. Furthermore if $n = 0$, then trivially $\gamma = \beta = f$, see Line 5. Hence, let us assume that $n \geq 1$. It follows that $\mathtt{fip}_k$ proceeds to construct finite intersection partitions for respective pairs of arguments γ_i and β_i. However, since $X \to f(\gamma_1, \ldots, \gamma_n)$ is conservative (see Remark A.4), it holds $\pi(\gamma_i) \leq \pi(X)$. At the same time, $\pi(\beta_i) < \pi(\beta)$; hence, by induction we can argue

that $\mathtt{fip}_k\, \gamma_i\, \beta_i$ constructs a proper intersection partition $\Pi(\gamma_i, \beta_i)$ for each $1 \leq i \leq n$, see Line 7. There exists therefore a unique term $\pi = f(\pi_1, \ldots, \pi_n) \in \mathtt{fip}_k\, X\, \beta$ such that $\delta \in L(\pi)$, see Line 7 and Line 17.

Conversely, take an arbitrary ground term $\delta = f(\delta_1, \ldots, \delta_n) \in L(\pi)$ for some $\pi \in \mathtt{fip}_k\, X\, \beta$ $(n \geq 0)$. By definition, $\mathtt{fip}_k$ proceeds to invoke itself on pairs of arguments γ and β where γ is the right-hand side of a production $X \to \gamma$ in $\mathcal{G}_k$, see Line 16, and returns the set-theoretic union of recursively obtained outcomes. There exists therefore some γ such that $\pi \in \mathtt{fip}_k\, \gamma\, \beta$. If $X \to \gamma$ is regular, then by induction, $\mathtt{fip}_k\, \gamma\, \beta$ constructs a FIP for both γ and β. Consequently, it holds $\delta \in L(\pi) \subset L(\gamma) \cap L(\beta) \subset L(X) \cap L(\beta)$. Assume therefore that $X \to \gamma$ is not regular, but instead self-referencing. As before, we cannot directly argue about $\mathtt{fip}_k\, \gamma\, \beta$ since the total potential of γ and β exceeds the potential of X and β. However, since $\pi \in \mathtt{fip}_k\, \gamma\, \beta$ and $\gamma \neq X$, we note that γ takes form $\gamma = f(\gamma_1, \ldots, \gamma_n)$, see Line 4. If f is a constant symbol, then certainly $\mathtt{fip}_k\, \gamma\, \beta$ outputs a proper FIP. Otherwise, $\mathtt{fip}_k$ proceeds to invoke itself recursively on respective pairs of arguments γ_i and β_i. Since $X \to f(\gamma_1, \ldots, \gamma_n)$ is conservative, we know that, by induction, $\mathtt{fip}_k\, \gamma_i\, \beta_i$ constructs finite intersection partitions $\Pi(\gamma_i, \beta_i)$ for all pairs γ_i and β_i. Certainly, $\delta_i \in L(\pi_i)$ for some $\pi_i \in \Pi(\gamma_i, \beta_i)$; hence $\delta \in L(\pi)$ where $\pi = f(\pi_1, \ldots, \pi_n) \in \mathtt{fip}_k\, \gamma\, \beta$. It follows that $\delta \in L(\pi) \cap L(\beta) \subset L(X) \cap L(\beta)$, which finishes the proof. ◀

▶ Remark A.5. Note that the termination of $\mathtt{fip}_k$ is based on the fact that all self-referencing productions of simple υ-reduction grammars are at the same time conservative. Indeed, $\mathtt{fip}_k$ does not terminate in the presence of non-conservative productions. Consider the non-conservative production $X \to f(f(X))$. Note that

$$\begin{aligned} \mathtt{fip}_k(f(f(X)), f(X)) &\to \mathtt{fip}_k(f(X), X) \\ &\to \mathtt{fip}_k(X, f(X)) \\ &\to \mathtt{fip}_k(f(f(X)), f(X)) \\ &\to \cdots \end{aligned} \tag{21}$$

B The construction of $(\mathcal{G}_n)_n$

Simple, verbose υ-reduction grammars satisfy the neat structural property of retaining closure width of generated terms. For that reason, we maintain both simplicity and verbosity as an invariant during the construction of $(\mathcal{G}_n)_n$.

Proof of Lemma 5.3. Suppose that neither $\chi = N$ nor $\chi = f(\alpha_1, \ldots, \alpha_m)$. Since $\mathcal{G}_n$ is simple, it follows that either $\chi = T$ or $\chi = G_k$ for some $0 \leq k \leq n$. However, due to the verbosity of $\mathcal{G}_n$ we know that $\chi \neq G_k$ and so it must hold $\chi = T$. Consider the following inductive family of terms:

$$\delta_1 = \underline{0}[\Uparrow (\uparrow)] \qquad \text{whereas} \qquad \delta_{n+1} = \underline{0}[\delta_n/]. \tag{22}$$

By construction, we note that $\delta_n \downarrow_n$. Let $s_1, \ldots, s_w$ be substitutions satisfying $s_i \in L(\sigma_i)$. Note that $\delta := \delta_{n+1}[s_1] \cdots [s_w] \in L(T[\sigma_1] \cdots [\sigma_w])$; hence, simultaneously δ reduces in n steps, as $\delta \in L(G_n)$, and in at least $n + 1$ steps, contradiction. ◀

The following result demonstrates a simple scheme of how φ-matchings can be exploited during the construction of new productions generating terms with specific head redexes.

Proof of Lemma 5.6. Let $t = (\lambda a)[s][s_1] \cdots [s_w] \downarrow_{n+1}$ where $w \geq 0$. Since t admits a head υ-redex, we note that $t \to t' = (\lambda(a[\Uparrow (s)]))\, [s_1] \cdots [s_w] \downarrow_n$. By assumption, $\mathcal{G}_n$ is simple,

hence there exists a unique production $G_n \to \gamma$ in $\mathcal{G}_n$ such that $t' \in L(\gamma)$. Consider the set Δ^n_φ. Since $G_n \to \gamma$ is the unique production satisfying $t' \in L(\gamma)$, it follows that for each production $G_n \to \gamma'$ in $\mathcal{G}_n$ such that $\gamma' \neq \gamma$ and all $\pi \in \Delta_\varphi(\gamma')$ it holds $t' \notin L(\pi)$. Let us therefore focus on the set $\Delta_\varphi(\gamma)$ of φ-matchings limited to γ.

By assumption, $\mathcal{G}_n$ is not only simple but also verbose. Consequently, we know that γ retains the closure width of generated terms, see Lemma 5.4. It follows that γ has closure width w and takes the form $\gamma = \chi[\tau_1]\cdots[\tau_w]$. Certainly, $t' \in L(\varphi_{\langle w\rangle})$. Moreover, $\Delta_\varphi(\gamma) = \Pi(\gamma, \varphi_{\langle w\rangle})$. There exists therefore a unique $\pi \in \Pi(\gamma, \varphi_{\langle w\rangle})$ such that $t' \in L(\pi)$. Given the fact that the head of $\varphi_{\langle w\rangle}$ is equal to $\varphi = \lambda(T[\Uparrow (S)])$ we note that π must be of form $\pi = \lambda(\alpha[\Uparrow (\sigma)])[\sigma_1]\cdots[\sigma_w]$. However, since $t' = (\lambda a[\Uparrow (s)][s_1]\cdots[s_w]) \in L(\pi)$ it also means that $a \in L(\alpha)$, $s \in L(\sigma)$, and $s_i \in L(\sigma_i)$ for all $1 \leq i \leq w$. Consequently, $t \in L((\lambda\alpha)[\sigma][\sigma_1]\cdots[\sigma_w])$ as required.

Conversely, let $\pi = \lambda(\alpha[\Uparrow (\sigma)])[\sigma_1]\cdots[\sigma_w]$ be the unique term in the φ-matching family Δ^n_φ such that $t \in L((\lambda\alpha)[\sigma][\sigma_1]\cdots[\sigma_w])$. Note that $a \in L(\alpha)$, $s \in L(\sigma)$, and $s_i \in L(\sigma_i)$ for all $1 \leq i \leq w$. Since t has a head υ-redex, after a single reduction step t reduces to $t' = \lambda(a[\Uparrow (s)])[s_1]\cdots[s_w] \in L(\pi)$. By construction of Δ^n_φ, it means that there exists a production $G_n \to \gamma$ in $\mathcal{G}_n$ such that $L(\pi) \subset L(\gamma)$ and hence $t' \downarrow_n$. Certainly, it follows that $t \downarrow_{n+1}$. ◀

C Quantitative analysis of $(\mathcal{G}_n)_n$

Techniques of analytic combinatorics provide systematic means of investigating various discrete structures through a direct analysis of their corresponding generating functions [12, 13]. Although the presented analysis is quite straightforward, it requires a fair amount of background knowledge. We refer the unfamiliar reader to Flajolet and Sedgewick's excellent textbook [13] for a thorough introduction to the subject.

Proof of Proposition 6.4. Let $G_k \to \gamma$ be a regular production in $\mathcal{G}_k$. Since by construction $\mathcal{G}_k$ is simple, we know that $\gamma \in \mathcal{T}_\mathcal{F}(\mathcal{N} \cup \{G_0, \ldots, G_{k-1}\})$. Following symbolic methods [13, Part A, Symbolic Methods] we can therefore convert each non-terminal $X \in \mathcal{N} \cup \{G_0, \ldots, G_{k-1}\}$ occurring in γ into an appropriate generating function $X(z)$. Likewise, we can convert each function symbol occurrence f into an appropriate monomial z, see Figure 5. Finally, we group respective monomials together, and note that the generating function $G_\gamma(z)$ corresponding to γ takes the form (15). Respective exponents denote the number of occurrences of their associated symbols.

Consider the remaining self-referencing productions $G_k \to \delta$. Again, since $\mathcal{G}_k$ is simple, we know that δ takes the form λG_k, $G_0 G_k$ or (symmetrically) $G_k G_0$. And so, as each $X \in \mathcal{N}_n$ is unambiguous in $\mathcal{G}_k$, by symbolic methods, it follows that $G_k(z)$ satisfies the following functional equation:

$$G_k(z) = zG_k(z) + 2zG_0(z)G_k(z) + \sum_{G_k \to \gamma \in \Phi_k} G_\gamma(z). \tag{23}$$

Note that as no $G_\gamma(z)$ references the left-hand side $G_k(z)$, equation (23) is in fact linear in $G_k(z)$. Furthermore, as $G_0(z) = L_\infty(z)$ we finally obtain the requested form of $G_k(z)$, see (14). ◀

Equipped with Proposition 6.4 we are now ready to prove the main quantitative result of the current paper.

Proof of Theorem 6.5. We claim that for each $k \geq 1$ the generating function $G_k(z)$ can be represented as $G_k(z) = \sqrt{1-4z}P(z) + Q(z)$ where both $P(z)$ and $Q(z)$ are functions analytic in the disk $|z| < \frac{1}{4} + \varepsilon$ for some positive ε. The asserted asymptotic estimate follows then as a straightforward application of algebraic singularity analysis, see Proposition 6.1.

We start with showing that each $G_k(z)$ includes a summand in form of $\sqrt{1-4z}\,\overline{P}(z)+\overline{Q}(z)$ such that both $\overline{P}(z)$ and $\overline{Q}(z)$ are analytic in a large enough disk containing (properly) $|z| < \frac{1}{4}$. Afterwards, we argue that no summand has singularities in $|z| < \frac{1}{4}$. Standard closure properties of analytic functions with single dominant, square-root type singularities guarantee the required representation of $G_k(z)$.

Let $\varphi = N$ be the template corresponding to the (RVar) rule, see Table 1. Note that since $\mathcal{G}_0$ includes the production $G_0 \to N$, the set of φ-matchings Δ_φ^0 consists of the single term N. Hence, due to the respective production construction, it means that $G_1 \to (\mathsf{S}\,N)[T/]$ is a production of $\mathcal{G}_1$, cf. Example 5.10. Moreover, as a consequence of the construction associated with the (FVar) rule, for each $G_k \to \gamma$ in $\mathcal{G}_k$ there exists a production $G_{k+1} \to \underline{\mathsf{0}}[\gamma/]$ in the subsequent grammar $\mathcal{G}_{k+1}$. And so, each $\mathcal{G}_k$ includes among its productions one production in form of

$$G_k \to \underbrace{\underline{\mathsf{0}}[\underline{\mathsf{0}}[\ldots\underline{\mathsf{0}}}_{k-1 \text{ times}}[(\mathsf{S}\,N)[T/]/]\ldots/]/] \tag{24}$$

Denote the right-hand side of the above production as γ. Note that the associated generating function $G_\gamma(z)$, cf. (23), must therefore take form

$$G_\gamma(z) = z^{\zeta(\gamma)}T(z)N(z) \qquad \text{and so} \qquad G_\gamma(z) = \sqrt{1-4z}\,\overline{P}(z) + \overline{Q}(z) \tag{25}$$

where both $P(z)$ and $Q(z)$ are analytic in $|z| < \frac{1}{4} + \varepsilon$ for some (determined) $\varepsilon > 0$.

In order to show that no production admits a corresponding generating function with singularities in the disk $|z| < \frac{1}{4}$ we note that the single dominant singularity of $G_0(z)$, and so at the same time $L_\infty(z)$, is equal to the smallest positive real root ρ of $1-3z-z^2-z^3$ which satisfies $\frac{1}{4} < \rho \approx 0.295598$, see Proposition 6.2. Due to the form of basic generating functions corresponding to T, S and N, see Proposition 6.3, we further note that other singularities must lie on the unit circle $|z| = 1$. And so, each $G_k(z)$ admits the asserted form

$$G_k(z) = \sqrt{1-4z}P(z) + Q(z) \tag{26}$$

for some functions analytic in a disk $|z| < \frac{1}{4} + \varepsilon$. ◀

Deriving an Abstract Machine for Strong Call by Need

Małgorzata Biernacka
Institute of Computer Science, University of Wrocław, Poland
mabi@cs.uni.wroc.pl

Witold Charatonik
Institute of Computer Science, University of Wrocław, Poland
wch@cs.uni.wroc.pl

Abstract

Strong call by need is a reduction strategy for computing strong normal forms in the lambda calculus, where terms are fully normalized inside the bodies of lambda abstractions and open terms are allowed. As typical for a call-by-need strategy, the arguments of a function call are evaluated at most once, only when they are needed. This strategy has been introduced recently by Balabonski et al., who proved it complete with respect to full β-reduction and conservative over weak call by need.

We show a novel reduction semantics and the first abstract machine for the strong call-by-need strategy. The reduction semantics incorporates syntactic distinction between strict and non-strict let constructs and is geared towards an efficient implementation. It has been defined within the framework of generalized refocusing, i.e., a generic method that allows to go from a reduction semantics instrumented with context kinds to the corresponding abstract machine; the machine is thus correct by construction. The format of the semantics that we use makes it explicit that strong call by need is an example of a hybrid strategy with an infinite number of substrategies.

2012 ACM Subject Classification Theory of computation → Lambda calculus; Theory of computation → Operational semantics

Keywords and phrases abstract machines, reduction semantics

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.8

Funding This research is supported by the National Science Centre of Poland, under grant number 2014/15/B/ST6/00619.

1 Introduction

Call by need, or *lazy evaluation*, is a strategy to evaluate λ-terms based on two principles: first, evaluation of an expression should be delayed until its value is needed; second, the arguments of a function call should be evaluated at most once [32]. Lazy evaluation can be considered as an optimization of call-by-name evaluation where the computation of arguments is delayed but its results are not reused. A model implementation of lazy evaluation is often given in the form of an abstract machine which typically includes a store to facilitate memoization of intermediate results, as in the well-known STG machine of Peyton Jones used in the Haskell compiler [26]. On the other hand, theoretical studies of lazy evaluation stem from two canonical approaches: a store-based natural semantics of Launchbury [24, 28], and a storeless, purely syntactic account of Ariola et al. [8, 25, 17, 21]. Lazy evaluation is an example of a strategy realizing *weak reduction*, which is standard in functional programming languages, where all λ-abstractions are considered to be values; consequently, evaluation of a term always stops after reducing it to a λ-abstraction. In contrast, *strong reduction* continues to reduce inside the bodies of λ-abstractions until a full β-normal form is reached; consequently, it must reduce inside substitutions and it must be able to evaluate open terms. Strong reduction and the corresponding reduction strategies have been gaining more

© Małgorzata Biernacka and Witold Charatonik;
licensed under Creative Commons License CC-BY
4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019).
Editor: Herman Geuvers; Article No. 8; pp. 8:1–8:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

attention due to the development of proof assistants based on dependent types, such as Agda or Coq, whose implementation requires full term normalization for type-checking [23, 14]. In particular, the current version of the Coq proof assistant [30] employs an abstract machine that uses a lazy strategy to fully normalize terms, but the strategy has not been studied formally. Recently, Balabonski et al. proposed a strong call-by-need reduction strategy as a theoretical foundation for implementations of strong call by need [10]. Their strategy is proved to be complete with respect to β-reduction in the λ-calculus and conservative over the weak call-by-need strategy. Even though it has good theoretical properties, it is not clear how the strategy can be efficiently implemented, because it lacks operational account.

The goal of this paper is twofold: to contribute to the study of strong call by need by presenting a novel reduction semantics and an abstract machine for the strategy described in [10], and to showcase the existing framework of generalized refocusing used to inter-derive the two, quite complex, semantic artefacts. To this end, we first give a proper formulation of reduction semantics for the strategy, one that fits the framework and thus directly enables its implementation, and which can be seen as an operationalized variant of Balabonski et al.'s semantics. Second, we derive an abstract machine from the strategy, by means of the refocusing procedure [12] which takes as input a reduction semantics satisfying mild syntactic conditions and produces a lower-level specification that is provably correct with respect to the reduction semantics. The procedure is implemented in Coq and is fully automatic, once the syntactic conditions are proved.

The reduction semantics we present is inspired by previous work on weak [9, 19] and strong [10] call-by-need strategies. In particular, it builds on the concepts from [10] and incorporates syntactic distinction between strict and non-strict let constructs from [19], and it does not introduce an explicit store. However, in contrast to [10], we avoid collecting non-local information in the process of decomposition of terms (such as traversing a term to identify all its needed variables), but rather we thread the required information throughout. We prove that our version of the reduction semantics is adequate with respect to that from [10]; the formal correspondence between the two can be found in Section 6. The reduction semantics is presented in a format recently developed in [12], based on contexts instrumented with kinds that carry extra information. This format makes it explicit that strong call by need is an example of a hybrid strategy with an infinite number of substrategies (which equals the number of nonterminal symbols in the grammar).

Since the strong call-by-need semantics is quite sophisticated, as a warm-up we show a reduction semantics and an abstract machine for weak call by need, which is much simpler and presented in the same framework of generalized refocusing.

The rest of the paper is organized as follows. In Section 2 we recall the semantic formats that we use throughout the paper. In Section 3 we show the reduction semantics for the weak call-by-need strategy due to Danvy and Zerny, and we derive the corresponding abstract machine in the framework of generalized refocusing. In Section 4 we present a novel formulation of a reduction semantics for the strong call-by-need strategy, in Section 5 we discuss the derived abstract machine for this strategy. In Section 6 we show that our semantics is equivalent to that of Balabonski et al. In Section 7 we discuss the closest related work, and we conclude in Section 8.

2 Preliminaries

A *reduction semantics* is a kind of small-step operational semantics, where the positions in a term that can be rewritten are explicitly defined by reduction contexts, rather than implicit in inference rules [20]. By a *context* we mean a term with exactly one occurrence of

a variable called *a hole* and denoted $\square$. For a given term t and a context C, by $C[t]$ we denote the result of *plugging* t into C, i.e., the term obtained by substituting the hole in C with t. We say that a pair $\langle C, t\rangle$ is a *decomposition* of the term $C[t]$ into the context C and the term t. The set of reduction contexts in a reduction semantics is often defined in form of a grammar of contexts [18]. A *contraction* relation $\rightharpoonup$ of a reduction semantics specifies atomic computation steps, typically given by a set of rewriting rules. Terms that can be rewritten by $\rightharpoonup$ are called *redices* and those produced by it – *contracta*. In a reduction semantics, the *reduction relation* $\to$ is defined as the compatible closure of the contraction: a term t reduces in one step to t' if it can be decomposed into a redex r in an evaluation context E, that is $t = E[r]$, the redex r can be rewritten in one step to t'' by one of the contraction rules, and t' is obtained by the recomposition of E and t'', that is $t' = E[t'']$. In Section 4 we use a more general definition of reduction that accounts for more complex strategies.

When considering programs as closed terms (i.e., terms without free variables), *evaluation* is defined as the reflexive-transitive closure of the reduction relation (written $\to^*$), and *values* are expected results of computation, chosen from the set of all normal forms (other normal forms are often called *stuck terms*). All λ-abstractions are typically considered values and evaluation does not enter λ-bodies. More generally, and when we consider reduction of open terms, we often use the term *normalization* instead of evaluation, and *normal form* rather than value. In this paper, we consider open terms and we use these terms interchangeably (effectively, we treat all normal forms as values).

A *grammar of contexts* consists of a set N of nonterminal symbols, a starting nonterminal, a set S of variables denoting syntactic categories and a set of productions. Productions have the form $C \to \tau$ where $C \in N$ and τ is a term with free variables in $N \cup \{\square\} \cup S$, with exactly one occurrence of a variable from $N \cup \{\square\}$. If the grammar of reduction contexts encoding a strategy contains just one nonterminal symbol, we call this strategy *uniform*; intuitively, one always proceeds in the same way when decomposing a term into a reduction context and a redex. On the other hand, strategies that require multiple nonterminals in grammars are *hybrid*: it is necessary to use different substrategies for finding redices, one substrategy for each nonterminal symbol.

Figure 1 contains an example of a grammar with two nonterminals C, E (where C is the starting nonterminal) and syntactic categories t, n and v of terms, neutral terms and values. Figure 2 contains an example of a grammar with one nonterminal E and syntactic categories t, v of terms and values, and $E[x]$ is a succinct notation for the category of *needy* terms, i.e., terms decomposable into a variable in context (an explicit definition of needy terms will be given later in Figure 4).

$$
\begin{array}{llll}
C & ::= \square_C \mid \lambda x.C \mid E\,t \mid n\,C & \text{where} & t ::= \lambda x.\,t \mid x \mid t\,t, \\
E & ::= \square_E \mid E\,t \mid n\,C & & n ::= x \mid n\,v, \\
 & & & v ::= n \mid \lambda x.\,v
\end{array}
$$

$$(\beta\text{−contraction}) \quad (\lambda x.t_1)\,t_2 \rightharpoonup t_1[x \mapsto t_2]$$

Figure 1 Normal-order reduction semantics from [12].

Figure 1 shows the normal-order strategy, which is a hybrid strategy. This strategy normalizes a term to its full β-normal form (if it exists) by first evaluating it to its weak-head normal form with the call-by-name strategy, and only then reducing subterms of the resulting weak-head normal form with the same normal-order strategy. There are two substrategies,

one for each nonterminal symbol in the grammar. The E substrategy corresponds to call by name and reduces to weak-head normal form; the C substrategy allows reduction in bodies of λ-abstractions and in arguments to neutral terms. Each substrategy comes with its own kind of hole – the subscript indicates the kind of context that can be built inside the hole. In other words, if we want to extend a reduction context with a hole of kind k by plugging another context in it, this new context has to be derivable from the nonterminal k. The contraction rule (standard β-reduction) is here common to both substrategies, but in Section 4 we show that the contraction relation may be parameterized by kinds.

Abstract machines abound in the literature. They may serve as theoretical artefacts that facilitate reasoning about the strategy and the execution of programs, or provide the basis for further transformations and optimizations in a principled way, possibly using known, off-the-shelf techniques. Intuitively, abstract machines are just another form of operational semantics, only defined at a lower level of abstraction. They typically provide reasonably precise and efficient models of implementation. Ideally, each step of an abstract machine should be done in constant time, which usually can be achieved by rewriting only topmost symbols of machine configurations. Therefore complex operations such as finding a redex in a term or substituting a term for a variable in another term should be divided into smaller steps, making explicit the process of decomposition of terms. It is nontrivial how to achieve this atomicity when the strategy requires non-local information to proceed – one typically needs to thread some extra information in machine configurations. The strong call-by-need strategy is an example where this happens and in this paper we show how we solve this problem for finding redices. However, we do not deal with the decomposition of the contraction rules into atomic steps, which is an orthogonal issue. In fact, decomposition of contraction can also be handled by the refocusing methodology, as witnessed by previous work deriving environment-based abstract machines from calculi of closures by refocusing [13]. We leave it for future work.

3 Weak call by need

As a gentle introduction we first review the simpler weak call-by-need strategy and discuss some of the concepts we later extend to the strong case. We also summarize the main idea behind the refocusing procedure that allows to derive an abstract machine from a reduction semantics.

3.1 Reduction strategy

There is a wide range of theoretical studies of lazy evaluation in the λ-calculus, presenting different semantic formats of the weak call-by-need strategy. Here we recall one: Danvy and Zerny's "revised storeless reduction semantics", which is most relevant to our work and can be seen as the basis for our strong variant of the call-by-need reduction semantics. This strategy was derived in [19] from the standard call-by-need reduction for the λ_{let} calculus common to Ariola, Felleisen, Maraist, Odersky and Wadler [8, 9, 25] as part of a bigger picture connecting the various approaches to the weak call-by-need strategy.

The strategy is presented in Figure 2. The grammar of terms extends lambda terms with two forms of let-constructors declaring denotables. A *strict* let expression $\texttt{let}\ x := t_0\ \texttt{in}\ t_1$ makes it syntactically explicit that the variable x is *needed* in the let-body t_1 and its value is still not known. Informally, we say that the variable x is needed in t_1 whenever the first thing to do when evaluating t_1 is to establish the value of x, i.e., when t_1 can be uniquely decomposed as $t_1 = E[x]$, where E is a reduction context. In a non-strict version

Syntax:

$$
\begin{array}{rcl}
\text{(terms)} & t & ::= x \mid \lambda x.t \mid t\,t \mid \texttt{let}\ x = t\ \texttt{in}\ t \mid \texttt{let}\ x := t\ \texttt{in}\ E[x] \\
(\lambda\text{-values}) & v & ::= \lambda x.\, t \\
\text{(answer contexts)} & A & ::= \Box \mid \texttt{let}\ x = t\ \texttt{in}\ A \\
\text{(evaluation contexts)} & E & ::= \Box \mid E\,t \mid \texttt{let}\ x = t\ \texttt{in}\ E \mid \texttt{let}\ x := E\ \texttt{in}\ E[x] \\
\text{(redices)} & r & ::= A[v]\,t \mid \texttt{let}\ x := A[v]\ \texttt{in}\ E[x] \mid \texttt{let}\ x = t\ \texttt{in}\ E[x]
\end{array}
$$

Contraction rules:

$$
\begin{array}{lrcl}
(1) & A[\lambda x.t]\,t_1 & \rightharpoonup & A[\texttt{let}\ x = t_1\ \texttt{in}\ t] \\
(2) & \texttt{let}\ x = t\ \texttt{in}\ E[x] & \rightharpoonup & \texttt{let}\ x := t\ \texttt{in}\ E[x] \\
(3) & \texttt{let}\ x := A[v]\ \texttt{in}\ E[x] & \rightharpoonup & A[\texttt{let}\ x = v\ \texttt{in}\ E[v]]
\end{array}
$$

Figure 2 The revised call-by-need λ_{let} calculus from [19].

$\texttt{let}\ x = t_0\ \texttt{in}\ t_1$ the variable x is possibly not needed or its value is already known. In other words, evaluation of a strict expression $\texttt{let}\ x := t_0\ \texttt{in}\ t_1$ starts with evaluation of t_0 while evaluation of a non-strict expression $\texttt{let}\ x = t_0\ \texttt{in}\ t_1$ starts with evaluation of t_1.

In Danvy and Zerny's setting only closed terms are considered, values are arbitrary λ-abstractions and the answers produced by evaluation are values possibly wrapped in a number of let bindings – the answers are represented here as values plugged in answer contexts A. Throughout this paper we assume the variable convention, i.e., that all the bound and free variable names are pairwise distinct. The evaluation contexts E encode the strategy; in an application we look for a redex in the operator position, and the strategy for the two let constructs described informally above is encoded in the last two context constructors.

Let us briefly discuss the contraction rules. Rule (1) implements delayed computation: we delay the evaluation of the actual parameter t_1 and instead start the computation of the body t. This is because the contractum here is a non-strict let expression, where, by the third production in the grammar of evaluation contexts, a reduction context is sought in the subterm t. Rule (2) states that the computation of x can no longer be delayed. Rule (3) implements memoization: when the value of the term bound to x is known, it not only replaces x in the current context, but also it is stored in the answer substitution. The let construct is no longer strict because we do not know if x will be needed again.[1]

▶ **Example 3.1.** To observe the benefits of the call-by-need strategy we should look at an evaluation of a term of the form $f\,t\,s$ where the terms t and s require some computation and f contains two formal parameters: one that occurs at least twice and the other that does not occur in the body. The simplest case is $f = \lambda x.\lambda y.x\,x$ and $t = \mathsf{id}\,\mathsf{id}$ where $\mathsf{id} = \lambda z.z$; a definition of s is not relevant. Figure 3 shows this evaluation. In each step the redex is underlined and the relevant contexts are marked with square brackets. In steps $(i) - (ii)$ the computations of x and y are delayed; in step (iii) x is needed. In steps $(iv) - (vi)$ the value of the term bound to x is computed and in step (vii) it is memoized. In step (x) x is needed again and in step (xi) the computed value is reused thus avoiding the second computation of t. Step (xii) finishes the computation in the body of f. The variable y is never needed, so the value of s is not computed.

[1] Danvy and Zerny also include a fourth rule which is a short-cut of rules (2) and (3) in the case when t is a value. We do not consider it here since it does not constitute a different contraction step but can be seen as an optimization of the reduction sequence.

$$
\begin{array}{rcl}
 & & [(\lambda x.\lambda y.x\,x)]\ (\mathsf{id}\,\mathsf{id})\ s \\
(i) & \overset{(1)}{\rightsquigarrow} & [\texttt{let}\ x = (\mathsf{id}\,\mathsf{id})\ \texttt{in}\ \lambda y.x\,x]\ s \\
(ii) & \overset{(1)}{\rightsquigarrow} & \texttt{let}\ x = (\mathsf{id}\,\mathsf{id})\ \texttt{in}\ \texttt{let}\ y = s\ \texttt{in}\ [x]\ x \\
(iii) & \overset{(2)}{\rightsquigarrow} & \texttt{let}\ x := ([\mathsf{id}]\ \mathsf{id})\ \texttt{in}\ \texttt{let}\ y = s\ \texttt{in}\ x\,x \\
(iv) & \overset{(1)}{\rightsquigarrow} & \texttt{let}\ x := (\texttt{let}\ z = \mathsf{id}\ \texttt{in}\ [z])\ \texttt{in}\ \texttt{let}\ y = s\ \texttt{in}\ x\,x \\
(v) & \overset{(2)}{\rightsquigarrow} & \texttt{let}\ x := (\texttt{let}\ z := [\mathsf{id}]\ \texttt{in}\ [z])\ \texttt{in}\ \texttt{let}\ y = s\ \texttt{in}\ x\,x \\
(vi) & \overset{(3)}{\rightsquigarrow} & \texttt{let}\ x := (\texttt{let}\ z = \mathsf{id}\ \texttt{in}\ [\mathsf{id}])\ \texttt{in}\ \texttt{let}\ y = s\ \texttt{in}\ [x]\ x \\
(vii) & \overset{(3)}{\rightsquigarrow} & \texttt{let}\ z = \mathsf{id}\ \texttt{in}\ \texttt{let}\ x = \mathsf{id}\ \texttt{in}\ \texttt{let}\ y = s\ \texttt{in}\ [\mathsf{id}]\ x \\
(viii) & \overset{(1)}{\rightsquigarrow} & \texttt{let}\ z = \mathsf{id}\ \texttt{in}\ \texttt{let}\ x = \mathsf{id}\ \texttt{in}\ \texttt{let}\ y = s\ \texttt{in}\ \texttt{let}\ z = x\ \texttt{in}\ [z] \\
(ix) & \overset{(2)}{\rightsquigarrow} & \texttt{let}\ z = \mathsf{id}\ \texttt{in}\ \texttt{let}\ x = \mathsf{id}\ \texttt{in}\ \texttt{let}\ y = s\ \texttt{in}\ \texttt{let}\ z := [x]\ \texttt{in}\ z \\
(x) & \overset{(2)}{\rightsquigarrow} & \texttt{let}\ z = \mathsf{id}\ \texttt{in}\ \texttt{let}\ x := [\mathsf{id}]\ \texttt{in}\ \texttt{let}\ y = s\ \texttt{in}\ \texttt{let}\ z := [x]\ \texttt{in}\ z \\
(xi) & \overset{(3)}{\rightsquigarrow} & \texttt{let}\ z = \mathsf{id}\ \texttt{in}\ \texttt{let}\ x = \mathsf{id}\ \texttt{in}\ \texttt{let}\ y = s\ \texttt{in}\ \texttt{let}\ z := [\mathsf{id}]\ \texttt{in}\ [z] \\
(xii) & \overset{(3)}{\rightsquigarrow} & \texttt{let}\ z = \mathsf{id}\ \texttt{in}\ \texttt{let}\ x = \mathsf{id}\ \texttt{in}\ \texttt{let}\ y = s\ \texttt{in}\ \texttt{let}\ z = \mathsf{id}\ \texttt{in}\ \mathsf{id}
\end{array}
$$

Figure 3 Evaluation of $(\lambda x.\lambda y.x\,x)\ (\mathsf{id}\,\mathsf{id})\ s$ in weak call-by-need.

3.2 Refocusing

Refocusing is a mechanical procedure for deriving abstract machines from reduction semantics. It was introduced in [18], formalized [29] in the Coq proof assistant, and recently generalized in [12]. The method was applied (both by hand and in Coq) to a number of reduction semantics, to derive new machines as well as to establish the connection between existing machines and their underlying reduction semantics [13, 21, 12].

A naive implementation of evaluation in a reduction semantics consists in repeating the following steps until the processed term is a normal form: (a) decompose the given term into a context and a redex, (b) contract the redex, and (c) recompose a new term by plugging the contractum in the context. Consider, for example, step (iv) in Figure 3. After contracting the redex $\mathsf{id}\,\mathsf{id}$, the contractum $\texttt{let}\ z = \mathsf{id}\ \texttt{in}\ z$ is wrapped in the context $\texttt{let}\ x := \Box\ \texttt{in}\ \texttt{let}\ y = s\ \texttt{in}\ x\,x$ in the recomposition phase of step (iv) and immediately unwrapped by removing the very same context in the decomposition phase of step (v).

Refocusing optimizes this naive implementation by avoiding the reconstruction of intermediate terms in a reduction sequence. To make this optimization work as an automatic procedure, the user must provide additional input – below we describe what is required by the current implementation. Figure 4 shows the essential part of this input in our implementation of weak call by need.

First, the user must define the set of values. The implementation requires all useful normal forms to be considered values, where by "useful" we mean that such a normal form put into some reduction context can potentially lead to further reduction. Under weak call by need, values are answers and needy terms, which are used in contraction (3) in Figure 2, and can be thought of as intermediate values. Their grammar is shown explicitly in Figure 4, parameterized by the needed variable (we use dash symbol - in place of variable when it is not relevant).

Second, the user must provide two functions, effectively defining the elementary contexts of the strategy. The first of them, denoted $\Downarrow$, takes a term t and tells what to do when this term is processed for the first time. The possible options are: decompose it (if t is

$$
\begin{array}{rll}
\text{(needy terms)} & n^x & ::= x \mid n^x\, t \mid \texttt{let}\ y = t\ \texttt{in}\ n^x \mid \texttt{let}\ y := n^x\ \texttt{in}\ n^y \\
\text{(answers)} & a & ::= \lambda x.\, t \mid \texttt{let}\ x = t\ \texttt{in}\ a
\end{array}
$$

$$
\begin{array}{rr}
 & [n^{\text{-}}]\, t \Uparrow \texttt{V} \\
 & [a]\, t \Uparrow \texttt{R} \\
x \Downarrow \texttt{V} & \texttt{let}\ x = t\ \texttt{in}\ [n^x] \Uparrow \texttt{R} \\
\lambda x.t \Downarrow \texttt{V} & \texttt{let}\ x = t\ \texttt{in}\ [n^y] \Uparrow \texttt{V}\ \text{if}\ x \neq y \\
t_1\, t_2 \Downarrow [t_1]\, t_2 & \texttt{let}\ x = t\ \texttt{in}\ [a] \Uparrow \texttt{V} \\
\texttt{let}\ x = t_1\ \texttt{in}\ t_2 \Downarrow \texttt{let}\ x = t_1\ \texttt{in}\ [t_2] & \texttt{let}\ x := [n^{\text{-}}]\ \texttt{in}\ n^x \Uparrow \texttt{V} \\
\texttt{let}\ x := t_1\ \texttt{in}\ t_2 \Downarrow \texttt{let}\ x := [t_1]\ \texttt{in}\ t_2 & \texttt{let}\ x := [a]\ \texttt{in}\ n^x \Uparrow \texttt{R}
\end{array}
$$

Figure 4 Input to the refocusing procedure for weak call-by-need.

decomposable); reduce it (if t is not decomposable and a redex, denoted `R`) or report a value (if t is not decomposable and a value, denoted `V`). In the decomposable case the function returns a decomposition into an elementary context and a subterm. (The `R` option is not used in the definition of $\Downarrow$ in the weak call-by-need strategy.)

The second function, denoted $\Uparrow$, tells how to process a term when we already know that it is a value. In such a case we have to take into account the context surrounding the term. Thus $\Uparrow$ takes an elementary context ec and a value v and informs how to process $ec[v]$; the options are as in the case of $\Downarrow$.

Third, in the case when the grammar of contexts contains overlapping productions, the user must also provide an order on these productions that prescribes which of the possible decompositions of a given term should be tried first (there are overlapping production in strong call by need, but not in weak call by need).

Finally, the user must prove that the defined semantics satisfies mild syntactic conditions described in detail in [12]. An example of such a condition is that the order on productions in the grammar is well-founded, which implies that all possible decompositions of a term are checked in a finite number of steps. The complete formalization of the weak call by need strategy can be found in the repository `http://bitbucket.org/pl-uwr/generalized_refocusing` in the file `examples/weak_cbnd.v`.

The resulting abstract machine is presented in Figure 5. We show it in a simplified form where some parts of configurations are omitted, e.g., the information about the kind of contexts, since there is only one kind and it does not affect the strategy.

The machine uses two kinds of configurations: an $\mathcal{E}$-configuration represents a term in a context, and the transition for these configurations correspond to the $\Downarrow$ function. A $\mathcal{C}$-configuration represents a value plugged in a context, and the transitions correspond to the $\Uparrow$ function. The evaluation of a term t starts in a configuration $\langle t, \square \rangle_{\mathcal{E}}$. In particular, the last two $\mathcal{E}$-transitions show the difference in the treatment of strict and non-strict let constructors. The third $\mathcal{C}$-transition makes it explicit when a let constructor should be treated as strict – exactly when the let-body is a term that needs the value of x. Since we admit open terms, it is possible that a let-body needs a different variable to proceed – in that case we treat such a term as an intermediate value (cf. the fourth $\mathcal{C}$-transition). The notation $n^{x \mapsto v}$ in the last transition stands for the result of substitution of the value v for the needed occurrence of the variable x in the needy term n^x.

$\langle x, C\rangle_{\mathcal{E}}$	$\longrightarrow$	$\langle C, x\rangle_{\mathcal{C}}$	
$\langle \lambda x.t, C\rangle_{\mathcal{E}}$	$\longrightarrow$	$\langle C, \lambda x.t\rangle_{\mathcal{C}}$	
$\langle t_1\, t_2, C\rangle_{\mathcal{E}}$	$\longrightarrow$	$\langle t_1, \Box\, t_2 \circ C\rangle_{\mathcal{E}}$	
$\langle \texttt{let}\ x = t_1\ \texttt{in}\ t_2, C\rangle_{\mathcal{E}}$	$\longrightarrow$	$\langle t_2, \texttt{let}\ x = t_1\ \texttt{in}\ \Box \circ C\rangle_{\mathcal{E}}$	
$\langle \texttt{let}\ x := t_1\ \texttt{in}\ t_2, C\rangle_{\mathcal{E}}$	$\longrightarrow$	$\langle t_1, \texttt{let}\ x := \Box\ \texttt{in}\ t_2 \circ C\rangle_{\mathcal{E}}$	
$\langle \Box\, t \circ C, n^{\bar{}}\rangle_{\mathcal{C}}$	$\longrightarrow$	$\langle C, n^{\bar{}}\, t\rangle_{\mathcal{C}}$	
$\langle \Box\, t_1 \circ C, A[\lambda x.t]\rangle_{\mathcal{C}}$	$\longrightarrow$	$\langle A[\texttt{let}\ x = t_1\ \texttt{in}\ t], C\rangle_{\mathcal{E}}$	
$\langle \texttt{let}\ x = t\ \texttt{in}\ \Box \circ C, n^x\rangle_{\mathcal{C}}$	$\longrightarrow$	$\langle \texttt{let}\ x := t\ \texttt{in}\ n^x, C\rangle_{\mathcal{E}}$	
$\langle \texttt{let}\ x = t\ \texttt{in}\ \Box \circ C, n^y\rangle_{\mathcal{C}}$	$\longrightarrow$	$\langle C, \texttt{let}\ x = t\ \texttt{in}\ n^y\rangle_{\mathcal{C}}$	for $x \neq y$
$\langle \texttt{let}\ x = t\ \texttt{in}\ \Box \circ C, a\rangle_{\mathcal{C}}$	$\longrightarrow$	$\langle C, \texttt{let}\ x = t\ \texttt{in}\ a\rangle_{\mathcal{C}}$	
$\langle \texttt{let}\ x := \Box\ \texttt{in}\ n^x \circ C, n^{\bar{}}\rangle_{\mathcal{C}}$	$\longrightarrow$	$\langle C, \texttt{let}\ x := n^{\bar{}}\ \texttt{in}\ n^x\rangle_{\mathcal{C}}$	
$\langle \texttt{let}\ x := \Box\ \texttt{in}\ n^x \circ C, A[v]\rangle_{\mathcal{C}}$	$\longrightarrow$	$\langle A[\texttt{let}\ x = v\ \texttt{in}\ n^{x \mapsto v}], C\rangle_{\mathcal{E}}$	

Figure 5 Abstract machine for weak call by need.

This machine differs from Danvy and Zerny's machine corresponding to the reduction semantics from Figure 2 in that it handles open terms and that it is not optimized, and not all steps can be executed in constant time. In particular, since terms of the form $A[t]$ are coerced to terms, each time we encounter a term of this form it will be decomposed from scratch, even though we know we could directly consider t in a context where all the surrounding let bindings are on the context stack. However, the structure of the resulting machine is amenable to off-the-shelf transformations leading to more efficient variants, e.g., following Danvy and Zerny's approach that allows to transform it into a store-based abstract machine [19].

Generalized refocusing. The distinction between uniform and hybrid strategies was first introduced in [22]. The refocusing procedure developed in [18] and formalized in [29] was limited to uniform strategies. Recently it has been generalized to hybrid strategies in [12] – the authors show several examples of strategies based on grammars with a few (usually two or three) nonterminal symbols, including the normal-order strategy shown in Figure 1.

Strong call by need is probably the first natural example of a reduction strategy with an unbounded number of nonterminals in the underlying grammar. In the remainder of the paper we present this strategy and the abstract machine derived by generalized refocusing.

4 Reduction semantics for strong call by need

We now present the reduction semantics for the strong call-by-need strategy. It is strongly inspired by the semantics from [10] in that it realizes exactly the same strategy, but is designed so that it fits in the refocusing framework, which facilitates the derivation of an abstract machine. The formal correspondence between our semantics and that of [10] is outlined in Section 6.

Terms. As in the case of weak call by need, the grammar of terms extends lambda terms with strict and non-strict let-constructors:

$$(\text{terms})\quad t \ ::= \ x \mid \lambda x.\, t \mid t\, t \mid \texttt{let}\ x = t\ \texttt{in}\ t \mid \texttt{let}\ x := t\ \texttt{in}\ t$$

The reduction semantics depends crucially on the notion of *frozen variables* introduced in [10]. A free variable is classified as frozen if it can never be substituted, either because it is not a let-bound variable (like x in $x\,t$), or if it is bound to a term that cannot become an answer (like x in $\texttt{let}\ x = z\,z\ \texttt{in}\ x\,t$). If a variable is not frozen, then we call it *active*. The status of variables depends on the context of evaluation: for example, when we consider the term $x\,t$ in the empty context, then x is frozen; but when the same term is plugged in the context $\texttt{let}\ x = \lambda z.z\ \texttt{in}\ \Box$ then x is no longer frozen. Intuitively, the latter context defrosts the variable and makes it active and substitutable.

In the process of evaluation we consider terms of particular shapes: structures S, normal forms N, and needy terms $N^{\cdot}_{\cdot}$. Structures and normal forms are defined by mutual recursion. Needy terms are parameterized by the variable they need and additionally by a context kind, which will be defined later. Whether a term falls into one of these categories depends on the status of its free variables, therefore each category is further parameterized by a set of variables containing exactly the frozen variables of a term.

The grammar of structures is shown in Figure 6. Informally, a structure is a normal form formed around a frozen variable, e.g., $\texttt{let}\ x = y\ \texttt{in}\ z\,(\lambda y.y)$ belongs to $S_{\{z\}}$ and is a structure formed around the frozen variable z. Structures here are almost equivalent to structures in [10], with the difference that here we parameterize them precisely with sets of frozen variables, while in [10] any superset of frozen variables is a good parameter.

$$\frac{}{x \in S_{\{x\}}} \qquad \frac{s \in S_\phi \quad n \in N_\psi}{s\,n \in S_{\phi \cup \psi}} \qquad \frac{s \in S_\phi \quad x \notin \phi}{\texttt{let}\ x = t\ \texttt{in}\ s \in S_\phi} \qquad \frac{s_1 \in S_\phi \quad s_2 \in S_\psi \quad x \in \psi}{\texttt{let}\ x := s_1\ \texttt{in}\ s_2 \in S_{(\phi \cup \psi) \setminus \{x\}}}$$

Figure 6 Grammar of structures.

Normal forms in N (with respect to strong call-by-need reduction) are either structures or irreducible terms built around a normal λ-abstraction in N^λ (see Figure 7). They can be seen as natural generalizations of weak call-by-need normal forms that arise when structures are introduced. The last two rules impose restrictions on the sets ϕ and ψ: first, x cannot be a member of ϕ because otherwise contexts $\texttt{let}\ x = t\ \texttt{in}\ \Box$ and $\texttt{let}\ x := s\ \texttt{in}\ \Box$ defrost it; second, n in the last rule occurs in a position of a needy term with needed variable x, so x must be a member of ψ.

$$\frac{s \in S_\phi}{s \in N_\phi} \qquad \frac{n \in N_\phi}{\lambda x.n \in N^\lambda_{\phi \setminus \{x\}}} \qquad \frac{n \in N^\lambda_\phi}{n \in N_\phi}$$

$$\frac{n \in N^\lambda_\phi \quad x \notin \phi}{\texttt{let}\ x = t\ \texttt{in}\ n \in N^\lambda_\phi} \qquad \frac{s \in S_\phi \quad n \in N^\lambda_\psi \quad x \in \psi \setminus \phi}{\texttt{let}\ x := s\ \texttt{in}\ n \in N^\lambda_{(\phi \cup \psi) \setminus \{x\}}}$$

Figure 7 Grammar of normal forms.

Finally, needy terms are terms uniquely decomposable into a reduction context and an active variable. Such terms can be seen as intermediate normal forms that arise in the normalization process when we try to establish if a given let-bound variable is needed. The grammar of needy terms is shown in Figure 8. It is important to note that the same term can be either a structure, or a needy term depending on the status of its free variables, e.g.,

$$\frac{}{x \in N^x_{k\cdot\emptyset}} \qquad \frac{n^x \in N^x_{\mathtt{E}\cdot\phi} \quad x \notin \phi}{n^x\, t \in N^x_{k\cdot\phi}} \qquad \frac{s \in S_\phi \quad n^x \in N^x_{\mathtt{C}\cdot\psi} \quad x \notin (\phi \cup \psi)}{s\, n^x \in N^x_{k\cdot\phi\,\cup\,\psi}}$$

$$\frac{n^x \in N^x_{k\cdot\phi} \quad x \neq y \quad y \notin \phi}{\mathtt{let}\ y = t\ \mathtt{in}\ n^x \in N^x_{k\cdot\phi}} \qquad \frac{s \in S_\psi \quad n^x \in N^x_{k\cdot\phi} \quad x \neq y \quad x \notin \psi}{\mathtt{let}\ y := s\ \mathtt{in}\ n^x \in N^x_{k\cdot((\phi\,\cup\,\psi)\,\setminus\,\{y\})}}$$

$$\frac{n^x \in N^x_{\mathtt{E}\cdot\phi}}{\mathtt{let}\ y := n^x\ \mathtt{in}\ n^y \in N^x_{k\cdot\phi}} \qquad \frac{n^x \in N^x_{\mathtt{C}\cdot\phi} \quad x \neq y}{\lambda y.n^x \in N^x_{\mathtt{C}\cdot\phi}}$$

Figure 8 Grammar of needy terms.

$x \in S_{\{x\}}$ (here x is frozen) and $x \in N^x_{k\cdot\emptyset}$ (here x is active). The latter term denotes the variable x, which is needed in the empty context (k denotes the kind of context) and active in terms of the form $\mathtt{let}\ x := t\ \mathtt{in}\ x$.

Reduction contexts. The strong call-by-need reduction strategy generalizes the weak call-by-need strategy – informally, it first tries to evaluate terms with the weak strategy and after reaching a weak value it attempts to normalize it further (i.e., inside a lambda abstraction or a neutral term). This pattern is analogous to the normal-order strategy, which can be adequately described using hybrid reduction contexts whose grammar defines the interconnection between the weak and the strong strategy (see Figure 1). To define strong contexts for normal order we need to use two kinds: `E` - for weak contexts (realizing call by name), and `C` - for strong contexts. When we consider call by need, we need to further instrument these kinds – they are parameterized by a set of frozen variables. Given a raw kind $k \in \{\mathtt{E}, \mathtt{C}\}$ and a set of frozen variables ϕ we write $k \cdot \phi$ for the kind k parameterized by ϕ. The union $\{x\} \cup \phi$ is abbreviated x, ϕ.

Reduction contexts are built from elementary contexts parameterized by two kinds; $EC^{k_1}_{k_2}$ denotes an elementary context of kind k_1 whose hole is of kind k_2. Figure 9 describes all the elementary contexts. In particular, $\lambda x.\Box$ is an elementary context of raw kind `C` (it is only available under the strong strategy), and when reducing under the lambda the same strong strategy is used with the λ-bound variable treated as frozen inside the body. In both the weak and the strong variant the context $\Box\, t$ can be used, but its hole always forces the weak strategy to be used inside (i.e., when we decompose the left-hand-side of an application we always use the weak strategy). The context $s\,\Box$ enforces strong reduction of the argument of an application – in case when the operand is a structure (therefore, irreducible and not creating a β-redex). The condition $\psi \subseteq \phi$ ensures that s is indeed a structure (and not a needy term) in the given context, see Example 4.1 at the end of this section.

The context $\mathtt{let}\ x = t\ \mathtt{in}\ \Box$ is used when first processing a let term, it considers the let-bound variable active inside the let-body – next we need to establish if it is needed there. The context $\mathtt{let}\ x := \Box\ \mathtt{in}\ n^x$ is available when we already know that x is needed in the let-body and it is necessary to compute the value of the term bound to the variable (using the weak strategy). Finally, the context $\mathtt{let}\ x := s\ \mathtt{in}\ \Box$ handles situations when the term bound to x does not evaluate to a weak λ-value but normalizes to a structure. In this case we go back to evaluating the let-body, this time with the let-bound variable treated as frozen because it will never be substituted (note that this is the second time the let-body will be decomposed but the decomposition will be different this time because the set of frozen variables has changed). Example 4.1 illustrates the last three rules.

$$\frac{x \notin \phi}{\lambda x.\Box \in EC^{\mathtt{C}\cdot\phi}_{\mathtt{C}\cdot x,\phi}} \qquad \frac{}{\Box\, t \in EC^{k\cdot\phi}_{\mathtt{E}\cdot\phi}} \qquad \frac{s \in S_\psi \quad \psi \subseteq \phi}{s\,\Box \in EC^{k\cdot\phi}_{\mathtt{C}\cdot\phi}}$$

$$\frac{x \notin \phi}{\mathtt{let}\ x = t\ \mathtt{in}\ \Box \in EC^{k\cdot\phi}_{k\cdot\phi}} \qquad \frac{}{\mathtt{let}\ x := \Box\ \mathtt{in}\ n^x \in EC^{k\cdot\phi}_{\mathtt{E}\cdot\phi}} \qquad \frac{s \in S_\psi \quad \psi \subseteq \phi \quad x \notin \phi}{\mathtt{let}\ x := s\ \mathtt{in}\ \Box \in EC^{k\cdot\phi}_{\mathtt{E}\cdot x,\phi}}$$

Figure 9 Elementary contexts for strong call by need.

General reduction contexts are composed from elementary ones with matching kinds:

$$\frac{}{\Box \in C^k_k} \qquad \frac{ec \in EC^{k_2}_{k_3} \quad c \in C^{k_1}_{k_2}}{ec \circ c \in C^{k_1}_{k_3}}$$

This is a representation of contexts inside-out (the hole of the context c matches the kind of the elementary context ec placed inside it).

Grammar. The grammar of reduction contexts contains nonterminals of the form $k \cdot \phi$, where $k \in \{\mathtt{C}, \mathtt{E}\}$ and ϕ is an arbitrary (finite) set of variables, with starting nonterminal $\mathtt{C} \cdot \emptyset$. The productions in this grammar have either the form $k \cdot \phi \to C[k' \cdot \psi]$ where C is an elementary context in $EC^{k\cdot\phi}_{k'\cdot\psi}$, or the form $k \cdot \phi \to \Box$. Note that since there are no restrictions on the number of variables in terms, the grammar contains an infinite number of productions.

Values. Because the strong call-by-need strategy mixes weak and strong normalization, the notion of value depends on the kind of the context.

$$\frac{n \in N_\phi \quad \phi \subseteq \psi}{n \in V_{\mathtt{C}\cdot\psi}} \qquad \frac{}{a \in V_{\mathtt{E}\cdot\psi}} \qquad \frac{s \in S_\phi \quad \phi \subseteq \psi}{s \in V_{\mathtt{E}\cdot\psi}} \qquad \frac{n^x \in N^x_{k\cdot\phi} \quad x \notin \psi \quad \phi \subseteq \psi}{n^x \in V_{k\cdot\psi}}$$

Strong values $V_{\mathtt{C}\cdot_}$ are all strong normal forms that match the kind. Weak values $V_{\mathtt{E}\cdot_}$ are answers, defined as in weak call by need (i.e., as lambda values in answer contexts, $a ::= A[v]$), and structures. The set of frozen variables dictated by the kind must be a superset of the set of frozen variables of the value (here again it ensures that s is indeed a structure in a given context). The strategy that we are describing works both for closed and for open terms – in the latter case we treat free variables as frozen. During normalization it happens that we produce intermediate values containing active variables – these are exactly the needy terms. The condition $x \notin \psi$ ensures that n^x is indeed a needy term, and not a structure under the given kind.

Contraction. Just like values, certain terms become redices (i.e., atomic reducible terms) only in a specific context. Therefore, the type of redices is parameterized by the kind.

$$\frac{}{(A[\lambda x.t])\, t' \rightharpoonup_{k\cdot\phi} A[\mathtt{let}\ x = t'\ \mathtt{in}\ t]}\ (\beta) \qquad \frac{n^x \in N^x_{k\cdot\phi}}{\mathtt{let}\ x := A[v]\ \mathtt{in}\ n^x \rightharpoonup_{k\cdot\phi} A[\mathtt{let}\ x = v\ \mathtt{in}\ n^{x\mapsto v}]}\ (\mathtt{lsv})$$

$$\frac{n^x \in N^x_{k\cdot\phi} \quad \phi \subseteq \psi}{\mathtt{let}\ x = t\ \mathtt{in}\ n^x \rightharpoonup_{k\cdot\psi} \mathtt{let}\ x := t\ \mathtt{in}\ n^x}\ (\mathtt{ls}) \qquad \frac{s \in S_{k\cdot\phi} \quad \phi \subseteq \psi}{\mathtt{let}\ x := s\ \mathtt{in}\ A[v] \rightharpoonup_{\mathtt{E}\cdot\psi} \mathtt{let}\ x = s\ \mathtt{in}\ A[v]}\ (\mathtt{lns})$$

The first three rules are generalizations of weak call-by-need contractions from Section 2, whereas the fourth one is added to handle open terms (structures). The first two rules

encode the usual call-by-need computation steps and they coincide with the contractions in [10], and the last two can be seen as "administrative" reductions that are responsible for marking/unmarking strict `lets`. The first rule creates a new let-binding whenever a lambda abstraction is applied to an argument (this binding will be processed lazily). The second rule is triggered in a situation when a needed let-bound variable has a value ready to be used in the let-body; the value is then substituted for the variable and the variable ceases to be needed. The third rule consists in marking the let-bound variable as needed when we know that the let-body is needy of this variable. The last rule is used to convert a strict let to an answer – when the variable is not really needed (it cannot happen if we start reducing with a term not containing strict lets).

Reduction. The reduction relation is defined as usual, with the restriction that the kind of contraction must match the kind of the hole in the reduction context: a term t reduces in one step to t' if it can be decomposed as $t = E[r]$ for some redex r and an evaluation context $E \in C^{-}_{k\cdot\phi}$, the redex r can be rewritten in one step to t'' by contraction $\rightharpoonup_{k\cdot\phi}$, and t' is obtained by the recomposition of E and t'', that is $t' = E[t'']$.

▶ **Example 4.1.** Consider an evaluation of a term of the form $\texttt{let}\ x = y\,y\ \texttt{in}\ x\,t$, where y is the only free (and frozen) variable, with the strong strategy $\texttt{C}\cdot\{y\}$. We first decompose the term to the context $\texttt{let}\ x = y\,y\ \texttt{in}\ \Box \in EC^{\texttt{C}\cdot\{y\}}_{\texttt{C}\cdot\{y\}}$ and the let-body $x\,t$. Here $x\,t \in N^{x}_{\texttt{C}\cdot\emptyset}$ is a needy term, so the input term is rewritten to $\texttt{let}\ x := y\,y\ \texttt{in}\ x\,t$, using the (`ls`) rule and contraction $\rightharpoonup_{\texttt{C}\cdot\{y\}}$. Since $y\,y$ is a structure in $S_{\{y\}}$ and $\{y\} \subseteq \{x, y\}$, the obtained term can be decomposed using the last rule in Fig. 9 to the context $\texttt{let}\ x := y\,y\ \texttt{in}\ \Box \in EC^{\texttt{C}\cdot\{y\}}_{\texttt{E}\cdot\{x,y\}}$ and the term $x\,t$, which we now visit for the second time. Note that the inclusion $\psi \subseteq \phi$ from Fig. 9 forces us to change the evaluation strategy here by adding x to the parameter set, so x becomes frozen. Since $x \in S_{\{x\}}$ and $\{x\} \subseteq \{x, y\}$, the third rule in Fig. 9 prescribes now to evaluate t using the strong strategy $\texttt{C}\cdot\{x, y\}$.

5 An abstract machine for strong call by need

In this section we present an abstract machine for strong call by need derived in the framework of generalized refocusing. The Coq development can be found in the repository `http://bitbucket.org/pl-uwr/generalized_refocusing` in the file `examples/strong_cbnd.v`. The machine (with some technical clutter removed) is presented in Figure 10. We assume implicit α-renaming of bound variables in order to avoid name clashes (cf. side conditions in transitions (3),(8),(21)).

The machine has two kinds of configurations. An $\mathcal{E}$-configuration of the form $\langle t, C, k, \phi\rangle_{\mathcal{E}}$ consists of a term, a surrounding context, and the kind of the context hole represented by the last two components of the configuration. In turn, a $\mathcal{C}$-configuration of the form $\langle C, v, k, \phi\rangle_{\mathcal{C}}$ consists of a context, a value plugged in this context, and the kind of the context hole. Because both contexts and values are parameterized by kinds, a configuration is considered correct if the last two components match the kind of the context component (in both $\mathcal{E}$ and $\mathcal{C}$-configurations) and of the value component (in $\mathcal{C}$-configurations). If we start the machine with a correct configuration, then all the configurations in the machine run are correct by construction.

The values computed by the machine (and used in $\mathcal{C}$-configurations) coincide with those in the reduction semantics. In Figure 10 we use notation n for arbitrary normal forms, s for structures, n^{λ} for lambda values, n^{x} and n^{y} for needy terms. Sometimes it is not obvious to

No.	Configuration		Result	Condition
1	t	$\longrightarrow$	$\langle t, \Box, \mathtt{C}, \emptyset\rangle_{\mathcal{E}}$	
2	$\langle t_1\, t_2, C, k, \phi\rangle_{\mathcal{E}}$	$\longrightarrow$	$\langle t_1, \Box\, t_2 \circ C_{k\cdot\phi}, \mathtt{E}, \phi\rangle_{\mathcal{E}}$	
3	$\langle \mathtt{let}\ x = t_1\ \mathtt{in}\ t_2, C, k, \phi\rangle_{\mathcal{E}}$	$\longrightarrow$	$\langle t_2, \mathtt{let}\ x = t\ \mathtt{in}\ \Box \circ C_{k\cdot\phi}, k, \phi\rangle_{\mathcal{E}}$	if $x \notin \phi$
4	$\langle \mathtt{let}\ x := t\ \mathtt{in}\ n^x, C, k, \phi\rangle_{\mathcal{E}}$	$\longrightarrow$	$\langle t, \mathtt{let}\ x := \Box\ \mathtt{in}\ n^x \circ C_{k\cdot\phi}, \mathtt{E}, \phi\rangle_{\mathcal{E}}$	
5	$\langle x, C, k, \phi\rangle_{\mathcal{E}}$	$\longrightarrow$	$\langle C, \mathbf{str}(x), k, \phi\rangle_{\mathcal{C}}$	if $x \in \phi$
6	$\langle x, C, k, \phi\rangle_{\mathcal{E}}$	$\longrightarrow$	$\langle C, \mathbf{nd}(x), k, \phi\rangle_{\mathcal{C}}$	if $x \notin \phi$
7	$\langle \lambda x.t, C, \mathtt{E}, \phi\rangle_{\mathcal{E}}$	$\longrightarrow$	$\langle C, \mathbf{ans}(\lambda x.t), \mathtt{E}, \phi\rangle_{\mathcal{C}}$	
8	$\langle \lambda x.t, C, \mathtt{C}, \phi\rangle_{\mathcal{E}}$	$\longrightarrow$	$\langle t, \lambda x.\Box \circ C_{\mathtt{C}\cdot\phi}, \mathtt{C}, x \cdot \phi\rangle_{\mathcal{E}}$	if $x \notin \phi$
9	$\langle \lambda x.\Box \circ C, n, \mathtt{C}, \phi\rangle_{\mathcal{C}}$	$\longrightarrow$	$\langle C, \lambda x.n, \mathtt{C}, \phi \setminus \{x\}\rangle_{\mathcal{C}}$	
10	$\langle \Box\, t \circ C_{k\cdot\psi}, \mathbf{ans}(A[\lambda x.r]), \mathtt{E}, \phi\rangle_{\mathcal{C}}$	$\longrightarrow$	$\langle A[\mathtt{let}\ x = t\ \mathtt{in}\ r], C, k, \psi\rangle_{\mathcal{E}}$	
11	$\langle \Box\, t \circ C_{k\cdot\psi}, \mathbf{nd}(n^y), \mathtt{E}, \phi\rangle_{\mathcal{C}}$	$\longrightarrow$	$\langle C, \mathbf{nd}(n^y\, t), k, \psi\rangle_{\mathcal{C}}$	if $y \notin \phi$
12	$\langle \Box\, t \circ C_{k\cdot\psi}, \mathbf{str}(s), \mathtt{E}, \phi\rangle_{\mathcal{C}}$	$\longrightarrow$	$\langle t, s\, \Box \circ C_{k\cdot\psi}, \mathtt{C}, \phi\rangle_{\mathcal{E}}$	
13	$\langle s\, \Box \circ C_{k\cdot\psi}, n, \mathtt{C}, \phi\rangle_{\mathcal{C}}$	$\longrightarrow$	$\langle C, \mathbf{str}(s\, n), k, \psi\rangle_{\mathcal{C}}$	
14	$\langle \mathtt{let}\ x = t\ \mathtt{in}\ \Box \circ C_{k\cdot\psi}, \mathbf{ans}(A[v]), \mathtt{E}, \phi\rangle_{\mathcal{C}}$	$\longrightarrow$	$\langle C, \mathbf{ans}(\mathtt{let}\ x = t\ \mathtt{in}\ A[v]), k, \psi\rangle_{\mathcal{C}}$	
15	$\langle \mathtt{let}\ x = t\ \mathtt{in}\ \Box \circ C_{k\cdot\psi}, \mathbf{nd}(n^x), k', \phi\rangle_{\mathcal{C}}$	$\longrightarrow$	$\langle \mathtt{let}\ x := t\ \mathtt{in}\ n^x, C, k, \psi\rangle_{\mathcal{E}}$	if $x \notin \phi$
16	$\langle \mathtt{let}\ x = t\ \mathtt{in}\ \Box \circ C_{k\cdot\psi}, \mathbf{nd}(n^y), k', \phi\rangle_{\mathcal{C}}$	$\longrightarrow$	$\langle C, \mathbf{nd}(\mathtt{let}\ x = t\ \mathtt{in}\ n^y), k, \psi\rangle_{\mathcal{C}}$	if $y \neq x, y \notin \phi$
17	$\langle \mathtt{let}\ x = t\ \mathtt{in}\ \Box \circ C_{k\cdot\psi}, \mathbf{str}(s), k', \phi\rangle_{\mathcal{C}}$	$\longrightarrow$	$\langle C, \mathbf{str}(\mathtt{let}\ x = t\ \mathtt{in}\ s), k, \psi\rangle_{\mathcal{C}}$	
18	$\langle \mathtt{let}\ x = t\ \mathtt{in}\ \Box \circ C_{k\cdot\psi}, \mathbf{lnf}(n^\lambda), \mathtt{C}, \phi\rangle_{\mathcal{C}}$	$\longrightarrow$	$\langle C, \mathbf{lnf}(\mathtt{let}\ x = t\ \mathtt{in}\ n^\lambda), k, \psi\rangle_{\mathcal{C}}$	
19	$\langle \mathtt{let}\ x := \Box\ \mathtt{in}\ n^x \circ C_{k\cdot\psi}, \mathbf{ans}(A[v]), \mathtt{E}, \phi\rangle_{\mathcal{C}}$	$\longrightarrow$	$\langle A[\mathtt{let}\ x = v\ \mathtt{in}\ n^{x\mapsto v}], C, k, \psi\rangle_{\mathcal{E}}$	
20	$\langle \mathtt{let}\ x := \Box\ \mathtt{in}\ n^x \circ C_{k\cdot\psi}, \mathbf{nd}(n^y), \mathtt{E}, \phi\rangle_{\mathcal{C}}$	$\longrightarrow$	$\langle C, \mathbf{nd}(\mathtt{let}\ x := n^y\ \mathtt{in}\ n^x), k, \psi\rangle_{\mathcal{C}}$	if $y \notin \phi$
21	$\langle \mathtt{let}\ x := \Box\ \mathtt{in}\ n^x \circ C_{k\cdot\psi}, \mathbf{str}(s), \mathtt{E}, \phi\rangle_{\mathcal{C}}$	$\longrightarrow$	$\langle n^x, \mathtt{let}\ x := s\ \mathtt{in}\ \Box \circ C_{k\cdot\psi}, \mathtt{E}, x \cdot \phi\rangle_{\mathcal{E}}$	if $x \notin \phi$
22	$\langle \mathtt{let}\ x := s\ \mathtt{in}\ \Box \circ C_{k\cdot\psi}, \mathbf{nd}(n^y), k', \phi\rangle_{\mathcal{C}}$	$\longrightarrow$	$\langle C, \mathbf{nd}(\mathtt{let}\ x := s\ \mathtt{in}\ n^y), k, \psi\rangle_{\mathcal{C}}$	if $y \notin \phi$
23	$\langle \mathtt{let}\ x := s\ \mathtt{in}\ \Box \circ C_{k\cdot\psi}, \mathbf{ans}(A[v]), \mathtt{E}, \phi\rangle_{\mathcal{C}}$	$\longrightarrow$	$\langle C, \mathbf{ans}(\mathtt{let}\ x = s\ \mathtt{in}\, A[v]), k, \psi\rangle_{\mathcal{C}}$	
24	$\langle \mathtt{let}\ x := s\ \mathtt{in}\ \Box \circ C_{k\cdot\psi}, \mathbf{str}(s'), k', \phi\rangle_{\mathcal{C}}$	$\longrightarrow$	$\langle C, \mathbf{str}(\mathtt{let}\ x := s\ \mathtt{in}\ s'), k, \psi\rangle_{\mathcal{C}}$	
25	$\langle \mathtt{let}\ x := s\ \mathtt{in}\ \Box \circ C_{k\cdot\psi}, \mathbf{lnf}(n^\lambda), \mathtt{C}, \phi\rangle_{\mathcal{C}}$	$\longrightarrow$	$\langle C, \mathbf{lnf}(\mathtt{let}\ x := s\ \mathtt{in}\ n^\lambda), k, \psi\rangle_{\mathcal{C}}$	
26	$\langle \Box, n, \mathtt{C}, \phi\rangle_{\mathcal{C}}$	$\longrightarrow$	n	

Figure 10 An abstract machine for strong call by need.

which category a given value falls; for example x in the configuration $\langle C, x, k, \phi\rangle_{\mathcal{C}}$ is either a structure or a needy term, depending on whether $x \in \phi$. To ease reading the transition rules, we attach tags to the different value constructors in the machine:

$$n ::= \mathbf{ans}(A[\lambda x.t]) \mid \mathbf{str}(s) \mid \mathbf{nd}(n^x) \mid \mathbf{lnf}(n^\lambda)$$

writing respectively $\langle C, \mathbf{str}(x), k, \phi\rangle_{\mathcal{C}}$ and $\langle C, \mathbf{nd}(x), k, \phi\rangle_{\mathcal{C}}$ instead of $\langle C, x, k, \phi\rangle_{\mathcal{C}}$.

An $\mathcal{E}$-configuration either decomposes a term by pushing a new elementary context on the existing context (in such a way that their kinds match) and proceeds with evaluation of a subterm, or it calls a $\mathcal{C}$-configuration if the term is a value. In particular, transition (4) prescribes that if a term is a strict let, then we need to evaluate (to weak value) the term t_1 bound to the variable. If the term happens to be a variable, it is a value but its status depends on the kind, more specifically on the set of frozen variables ϕ in the configuration: if the variable is not in this set, then it will be treated as active (cf. transitions (5) and (6)). A lambda abstraction is a value in a `E`-hole but it is further decomposed in a `C`-hole.

A $\mathcal{C}$-configuration dispatches on the context when a (matching) value is plugged in its hole. For example, transition (10) encodes β-contraction, transition (15) encodes the `ls`-contraction, transition (19) encodes the `lsv`-contraction, and transition (23) encodes the `lns`-contraction.

The machine correctly realizes the strong call-by-need strategy, it decomposes terms based on local information given in a configuration, but it still could be optimized in various directions, in particular, using insights from existing work on abstract machines for weak lazy evaluation [3, 19]. It is future work to check which of the transformations discussed there generalize to the strong case, in particular how to systematically obtain an efficient store-based machine.

6 Correctness

In this section we show how our reduction semantics relates to the Balabonski et al.'s. To this end, we need to introduce some of the notions they use in their work. In the following, we refer to the their language as Λ, to our language as Λ_s, and to standard lambda calculus as Λ_β.

Balabonski et al. do not distinguish between strict and non-strict let syntactically, they only have one form of let-construct. In order to discover the difference, they have to traverse the term to check if the let-bound variable is really needed, i.e., if it is in the set of non-garbage variables of the body. Non-garbage variables are defined as follows:

$$\begin{array}{rcl}
\mathtt{ngv}(x) & = & \{x\} \\
\mathtt{ngv}(\lambda x.t) & = & \mathtt{ngv}(t) \setminus \{x\} \\
\mathtt{ngv}(t_1\, t_2) & = & \mathtt{ngv}(t_1) \cup \mathtt{ngv}(t_2) \\
\mathtt{ngv}(t_2[x \backslash t_1]) & = & \mathtt{ngv}(t_2) \setminus \{x\} \cup \begin{cases} \mathtt{ngv}(t_1), & \text{if } x \in \mathtt{ngv}(t_2) \\ \emptyset, & \text{otherwise} \end{cases}
\end{array}$$

where the notation $t_2[x\backslash t_1]$ denotes an explicit substitution of t_1 for x in t_2.

In contrast, our intermediate language makes this distinction effective just as soon as decomposition of the term reveals it.

We define an erasure operation $|\cdot| : \Lambda_s \to \Lambda$:

$$\begin{array}{rcl} |x| & = & x \\ |\lambda x.t| & = & \lambda x.|t| \\ |t_1\, t_2| & = & |t_1|\, |t_2| \\ |\mathtt{let}\ x = t_1\ \mathtt{in}\ t_2| & = & |t_2|[x\backslash |t_1|] \\ |\mathtt{let}\ x := t_1\ \mathtt{in}\ t_2| & = & |t_2|[x\backslash |t_1|] \end{array}$$

This operation gives us a translation from Λ_s to Λ. A translation from Λ to Λ_s is trivial: every term in Λ is a term in Λ_s (modulo the notation; $t_2[x\backslash t_1]$ is represented as $\mathtt{let}\ x = t_1\ \mathtt{in}\ t_2$ in Λ).

We can show that the set of frozen variables of Λ_s-normal forms coincides with the non-garbage variables. All lemmas below have routine inductive proofs, which we omit here.

▶ **Lemma 6.1.** *For all Λ_s-normal forms $n \in N_\phi$ and for all variables $x \in \phi$, $x \in \mathtt{ngv}(|n|)$.*

The following two lemmas give a correspondence between normal forms in Λ and Λ_s. Here $\mathtt{N}_\phi$ denotes the set of normal terms (in Λ) under the set of frozen variables ϕ.

▶ **Lemma 6.2.** *For all Λ_s-normal forms $n \in N_\phi$, $|n| \in \mathtt{N}_\phi$ (it is a Λ-normal term).*

▶ **Lemma 6.3.** *For all Λ-normal forms $n \in \mathtt{N}_\psi$, there exist n_0 and $\phi \subseteq \psi$ such that $|n_0| = n$ and $n_0 \in N_\phi$.*

The next lemma states that needy terms correspond to Λ-contexts with holes filled with a designated variable. The notation $\mathtt{C}[\![x]\!]$ comes from [10] and denotes the variable x plugged in the context $\mathtt{C}$ that does not capture the variable (there are no abstractions or explicit substitutions that bind x in $\mathtt{C}$). Similarly, notation $\mathtt{E}_\phi$ (respectively, $\mathtt{E}^{@}_\phi$) is introduced in [10] and denotes evaluation contexts (respectively, inert evaluation contexts) under the set of frozen variables ϕ, both defined in [10].

▶ **Lemma 6.4.** *For every needy term $n^x \in N^x_{\mathtt{E}\cdot\phi}$ ($n^x \in N^x_{\mathtt{C}\cdot\phi}$, resp.) there exists a Λ-context $\mathtt{C} \in \mathtt{E}^{@}_\phi$ ($\mathtt{C} \in \mathtt{E}_\phi$, resp.) such that $|n^x| = \mathtt{C}[\![x]\!]$.*

In order to relate contexts, we need to introduce the conversion operator that transforms Λ_s-elementary contexts into Λ-contexts.

$$\begin{array}{rcl} |\lambda x.\Box|_c & = & \lambda x.\Box \\ |\Box\, t|_c & = & \Box\, |t| \\ |s\, \Box|_c & = & |s|\, \Box \\ |\mathtt{let}\ x = t\ \mathtt{in}\ \Box|_c & = & \Box[x\backslash |t|] \\ |\mathtt{let}\ x := \Box\ \mathtt{in}\ n^x|_c & = & \mathtt{C}[\![x]\!][x\backslash \Box] \quad \text{where } \mathtt{C}[\![x]\!] = |n^x| \\ |\mathtt{let}\ x := s\ \mathtt{in}\ \Box|_c & = & \Box[x\backslash |s|] \end{array}$$

Based on this definition and the composition of contexts in Λ, we can translate between Λ_s-contexts and Λ-contexts. Here $EC^{k\cdot\phi}_{_}$ is the union $\bigcup_{k',\psi} EC^{k\cdot\phi}_{k'\cdot\psi}$.

▶ **Lemma 6.5.** *For every elementary context $ec \in EC^{\mathtt{C}\cdot\phi}_{_}$ there is a Λ-context $|ec|_c \in \mathtt{E}_\phi$, and for every elementary context $ec \in EC^{\mathtt{E}\cdot\phi}_{_}$ there is a Λ-context $|ec|_c \in \mathtt{E}^{@}_\phi$ and such that*

$$\forall t, |ec[t]| = |ec|_c[t].$$

▶ **Lemma 6.6.** *For every Λ-context $\mathtt{C} \in \mathtt{E}_\phi$ ($\mathtt{C} \in \mathtt{E}^{@}_\phi$, resp.) there is a Λ_s-context $c \in C^{\mathtt{C}\cdot\psi}_{_}$ ($c \in C^{\mathtt{E}\cdot\psi}_{_}$, resp.) such that $\psi \subseteq \phi$ and $|c|_c = \mathtt{C}$.*

Having the ability to translate between both terms and contexts in the two languages, we can prove that Λ_s correctly simulates reductions in Λ. The reduction in Λ_s possibly uses more steps then the reduction in Λ because of switching between strict and non-strict versions of let constructs.

▶ **Definition 6.7.** *We say that a term $t \in \Lambda_s$ is proper if all its subterms of the form $\mathtt{let}\ x := t_1\ \mathtt{in}\ t_2$ are such that $t_2 = n^x$ for some needy term n^x, i.e., that strict `let`'s are correctly marked.*

▶ **Proposition 6.8.** *For all Λ-terms t_1 and t_2, if $t_1 \to_\Lambda t_2$ then there exists a proper $t_s \in \Lambda_s$ such that $t_1 \to^*_{\Lambda_s} t_s$ and $|t_s| = t_2$. Moreover, for all proper $t_s \in \Lambda_s, t \in \Lambda$, if $|t_s| = t$ then $t \to^*_{\Lambda_s} t_s$.*

▶ **Proposition 6.9.** *For all $t_1, t_2 \in \Lambda_s$, if $t_1 \to_{\Lambda_s} t_2$ then $|t_1| = |t_2|$ or $|t_1| \to_\Lambda |t_2|$. There is no infinite sequence $t_1 \to_{\Lambda_s} t_2 \to_{\Lambda_s} \ldots \to_{\Lambda_s} t_n \to_{\Lambda_s} \ldots$ such that $|t_1| = |t_n|$ for all n.*

As a consequence of the correct simulation result we obtain that Λ_s achieves the same normal forms as Λ.

▶ **Lemma 6.10.** *For all ϕ and for all $t, r \in \Lambda$ such that $t \to^*_\Lambda r$ and $r \in \mathtt{N}_\phi$,*

$$t \to^*_{\Lambda_s} r_s$$

for some $r_s \in N_\psi$ with $\psi \subseteq \phi$, and such that $|r_s| = r$.

▶ **Lemma 6.11.** *For all ϕ, and for all $t \in \Lambda, r_s \in \Lambda_s$ such that $t \to^*_{\Lambda_s} r_s$ and $r_s \in N_\phi$,*

$$t \to^*_\Lambda |r_s| \text{ holds and } |r_s| \in \mathtt{N}_\phi.$$

Now the completeness and conservativity results propagate from [10] to our setting. Here $(\cdot)^\diamond$ denotes the unfolding function defined in [10], which is a translation from Λ to Λ_β. The completeness result expresses that whenever a pure lambda term reduces to a normal form, the same normal form can be reached by the strong call-by-need strategy followed by unfolding the substitutions wrapping the obtained value.

▶ **Corollary 6.12** (Completeness). *For all lambda terms $t \in \Lambda_\beta$, if t reduces to a normal form r in Λ_β (in symbols, $t \to^*_{\Lambda_\beta} r$), then there exists a normal form r_s in Λ_s such that*

$$t \to^*_{\Lambda_s} r_s$$

and $|r_s|^\diamond = r$.

The conservativity result expresses that any strong call-by-need reduction has a weak call-by-need reduction as a prefix.

▶ **Corollary 6.13** (Conservativity). *For all lambda terms $t \in \Lambda_s$, if*

$$t = t_0 \to_{\Lambda_s} t_1 \to_{\Lambda_s} t_2 \to_{\Lambda_s} \ldots \to_{\Lambda_s} t_n$$

is a sequence of reductions ending with a normal form t_n, then there exists an $i \leq n$ such that $t_0 \to \ldots \to t_i$ is a reduction with weak call-by-need strategy.

7 Related work

Operational accounts for lazy evaluation are numerous, coming both from the practical and the theoretical considerations. The common implementation models include a canonical store-based abstract machine, where the store component is used to memoize the computed values and facilitate their reuse [26], as well as graph reduction machines devised on a principle of sharing of subgraphs representing argument terms [31]. On the other hand, theoretical investigations of call by need focus on establishing equational reasoning principles for lazy evaluation and various calculi have been developed for this purpose, with the canonical store-based natural semantics [24], and a storeless calculus based on let-constructs [8, 25]. These two worlds cross-fertilize, and there exist machines derived in an ad hoc manner from calculi, e.g., Sestoft's machine obtained from Launchbury's semantics [28].

A more principled approach to interderiving semantic artefacts has been advocated by Danvy and his collaborators; it consists in mechanizing derivations by identifying common transformation patterns that can be applied generically and provide guidance in the derivation process. In particular, they have used the functional correspondence to connect evaluators with abstract machines – including weak call by need [6, 7, 27], and the syntactic correspondence (based on refocusing) to connect reduction semantics and abstract machines [13].

Specific to call by need is an operational account of Danvy and Zerny who unify the various operational artefacts for lazy evaluation and provide a systematic approach to go from the canonical let-calculus of Ariola et al. through a series of refined reduction semantics and the corresponding abstract machines to a canonical store-based abstract machine (lazy Krivine machine) [19].

Strong reduction has been less studied operationally. The prominent examples of abstract machines for strong normalization include Crégut's abstract machine extending the Krivine machine [15, 16], which has later been derived more systematically by Nogueira and Garcia-Perez from the normal-order reduction strategy [22]; another variant has been devised using Linear Substitution Calculus [2] and useful sharing [1]. A different approach has been taken by Gregoire and Leroy whose normalization strategy uses call by value rather than call by name as a substrategy and was motivated by Coq implementation [23], and has later been refined as an instance of normalization by evaluation [14]. Efficient abstract machines for strong call by value have been recently devised by Accattoli et al. using Fireball Calculus [4, 5]. The only work on strong call by need seems to be Balabonski et al.'s strategy which we build on [10], recently extended to the Calculus of Inductive Constructions [11], focused on ensuring completeness of the strong call-by-need strategy.

The framework we work with is based on generalized refocusing that generalizes the syntactic correspondence used for weak strategies in that it allows for more complex strategies to be expressed (ones that can be seen as compositions of several substrategies), and the corresponding abstract machines to be derived [12].

8 Conclusion

We presented a systematic approach to defining reduction semantics for the call-by-need strategies, both weak and strong, in the framework of generalized refocusing. Within this framework we derived the corresponding abstract machines that are correct by construction. We observed that this approach is very effective – the requirements posed by the framework provide just enough structure and constraints to guide us in the process; in contrast, devising a machine from scratch would be much more a trial-and-error process, and quite tedious.

The derived machine is not optimized and thus not as effective as it might be. Our approach opens the possibility to systematically transform and optimize it. We leave it to the future work to further connect it to other semantic formats and models of implementation (in particular, a store-based abstract machine), in the spirit of the derivational approach of Danvy and Zerny [19], and to analyse its complexity as in the line of work of Accattoli [3].

References

1. Beniamino Accattoli. The Useful MAM, a Reasonable Implementation of the Strong λ-Calculus. In *Logic, Language, Information, and Computation - 23rd International Workshop, WoLLIC 2016, Puebla, Mexico, August 16-19th, 2016. Proceedings*, volume 9803 of *Lecture Notes in Computer Science*, pages 1–21. Springer, 2016. doi:10.1007/978-3-662-52921-8_1.
2. Beniamino Accattoli, Pablo Barenbaum, and Damiano Mazza. A Strong Distillery. In *Proceedings of APLAS*, volume 9458 of *Lecture Notes in Computer Science*, pages 231–250. Springer, 2015.
3. Beniamino Accattoli and Bruno Barras. Environments and the complexity of abstract machines. In Wim Vanhoof and Brigitte Pientka, editors, *Proceedings of the 19th International Symposium on Principles and Practice of Declarative Programming (PPDP'17), Namur, Belgium, October 09 - 11, 2017*, pages 4–16. ACM, 2017. doi:10.1145/3131851.3131855.
4. Beniamino Accattoli and Claudio Sacerdoti Coen. On the Relative Usefulness of Fireballs. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 141–155. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.23.
5. Beniamino Accattoli and Giulio Guerrieri. Implementing Open Call-by-Value. In Mehdi Dastani and Marjan Sirjani, editors, *Fundamentals of Software Engineering - 7th International Conference, FSEN 2017, Tehran, Iran, April 26-28, 2017, Revised Selected Papers*, volume 10522 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2017. doi:10.1007/978-3-319-68972-2_1.
6. Mads Sig Ager, Dariusz Biernacki, Olivier Danvy, and Jan Midtgaard. A Functional Correspondence between Evaluators and Abstract Machines. In Dale Miller, editor, *Proceedings of the Fifth ACM-SIGPLAN International Conference on Principles and Practice of Declarative Programming (PPDP'03)*, pages 8–19, Uppsala, Sweden, August 2003. ACM Press.
7. Mads Sig Ager, Olivier Danvy, and Jan Midtgaard. A Functional Correspondence between Call-by-Need Evaluators and Lazy Abstract Machines. *Inf. Process. Lett.*, 90(5):223–232, 2004. Extended version available as the research report BRICS RS-04-3.
8. Zena M. Ariola and Matthias Felleisen. The Call-By-Need lambda Calculus. *Journal of Functional Programming*, 7(3):265–301, 1997.
9. Zena M. Ariola, Matthias Felleisen, John Maraist, Martin Odersky, and Philip Wadler. The Call-by-Need Lambda Calculus. In Peter Lee, editor, *Proceedings of the Twenty-Second Annual ACM Symposium on Principles of Programming Languages*, pages 233–246, San Francisco, California, January 1995. ACM Press.
10. Thibaut Balabonski, Pablo Barenbaum, Eduardo Bonelli, and Delia Kesner. Foundations of strong call by need. *PACMPL*, 1(ICFP):20:1–20:29, 2017. doi:10.1145/3110264.
11. Pablo Barenbaum, Eduardo Bonelli, and Kareem Mohamed. Pattern Matching and Fixed Points: Resource Types and Strong Call-By-Need: Extended Abstract. In *PPDP*, pages 6:1–6:12. ACM, 2018.
12. Malgorzata Biernacka, Witold Charatonik, and Klara Zielinska. Generalized Refocusing: From Hybrid Strategies to Abstract Machines. In Dale Miller, editor, *2nd International Conference on Formal Structures for Computation and Deduction, FSCD 2017, September 3-9, 2017, Oxford, UK*, volume 84 of *LIPIcs*, pages 10:1–10:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.FSCD.2017.10.
13. Małgorzata Biernacka and Olivier Danvy. A Syntactic Correspondence between Context-Sensitive Calculi and Abstract Machines. *Theor. Comput. Sci.*, 375(1-3):76–108, 2007.

14 Mathieu Boespflug, Maxime Dénès, and Benjamin Grégoire. Full Reduction at Full Throttle. In *Proceedings of the First International Conference on Certified Programs and Proofs*, CPP'11, pages 362–377, Berlin, Heidelberg, 2011. Springer-Verlag. doi:10.1007/978-3-642-25379-9_26.

15 Pierre Crégut. An abstract machine for lambda-terms normalization. In Mitchell Wand, editor, *Proceedings of the 1990 ACM Conference on Lisp and Functional Programming*, pages 333–340, Nice, France, June 1990. ACM Press.

16 Pierre Crégut. Strongly Reducing Variants of the Krivine Abstract Machine. *Higher-Order and Symbolic Computation*, 20(3):209–230, 2007. A preliminary version was presented at the 1990 ACM Conference on Lisp and Functional Programming.

17 Olivier Danvy, Kevin Millikin, Johan Munk, and Ian Zerny. Defunctionalized Interpreters for Call-by-Need Evaluation. In Matthias Blume and German Vidal, editors, *Functional and Logic Programming, 10th International Symposium, FLOPS 2010*, number 6009 in Lecture Notes in Computer Science, pages 240–256, Sendai, Japan, April 2010. Springer.

18 Olivier Danvy and Lasse R. Nielsen. Refocusing in Reduction Semantics. Research Report BRICS RS-04-26, DAIMI, Department of Computer Science, Aarhus University, Aarhus, Denmark, November 2004. A preliminary version appeared in the informal proceedings of the Second International Workshop on Rule-Based Programming (RULE 2001), Electronic Notes in Theoretical Computer Science, Vol. 59.4.

19 Olivier Danvy and Ian Zerny. A Synthetic Operational Account of Call-by-need Evaluation. In *Proceedings of the 15th Symposium on Principles and Practice of Declarative Programming*, PPDP '13, pages 97–108, New York, NY, USA, 2013. ACM. doi:10.1145/2505879.2505898.

20 Matthias Felleisen, Robert Bruce Findler, and Matthew Flatt. *Semantics Engineering with PLT Redex*. The MIT Press, 1st edition, 2009.

21 Ronald Garcia, Andrew Lumsdaine, and Amr Sabry. Lazy evaluation and delimited control. In Benjamin C. Pierce, editor, *Proceedings of the Thirty-Fifth Annual ACM Symposium on Principles of Programming Languages*, pages 153–164. ACM Press, January 2009.

22 A García-Pérez and Pablo Nogueira. On the syntactic and functional correspondence between hybrid (or layered) normalisers and abstract machines. *Science of Computer Programming*, 95:176–199, 2014.

23 Benjamin Grégoire and Xavier Leroy. A compiled implementation of strong reduction. In Simon Peyton Jones, editor, *Proceedings of the 2002 ACM SIGPLAN International Conference on Functional Programming (ICFP'02)*, SIGPLAN Notices, Vol. 37, No. 9, pages 235–246, Pittsburgh, Pennsylvania, September 2002. ACM Press.

24 John Launchbury. A Natural Semantics for Lazy Evaluation. In Susan L. Graham, editor, *Proceedings of the Twentieth Annual ACM Symposium on Principles of Programming Languages*, pages 144–154, Charleston, South Carolina, January 1993. ACM Press.

25 John Maraist, Martin Odersky, and Philip Wadler. The Call-by-Need Lambda Calculus. *Journal of Functional Programming*, 8(3):275–317, 1998.

26 Simon L. Peyton Jones. Implementing Lazy Functional Languages on Stock Hardware: The Spineless Tagless G-Machine. *Journal of Functional Programming*, 2(2):127–202, 1992.

27 Maciej Piróg and Dariusz Biernacki. A systematic derivation of the STG machine verified in Coq. In Jeremy Gibbons, editor, *Proceedings of the 2010 ACM SIGPLAN Haskell Symposium (Haskell'10)*, pages 25–36, Baltimore, MD, September 2010. ACM Press.

28 Peter Sestoft. Deriving a Lazy Abstract Machine. *Journal of Functional Programming*, 7(3):231–264, May 1997.

29 Filip Sieczkowski, Małgorzata Biernacka, and Dariusz Biernacki. Automating derivations of abstract machines from reduction semantics: a generic formalization of refocusing in Coq. In Juriaan Hage and Marco T. Morazán, editors, *The 22nd International Conference on Implementation and Application of Functional Languages (IFL 2010)*, number 6647 in Lecture Notes in Computer Science, pages 72–88, Alphen aan den Rijn, The Netherlands, September 2010. Springer-Verlag.

30 The Coq Development Team. The Coq Proof Assistant, V. 8.7, 2018. URL: `https://github.com/coq/coq`.

31 David A. Turner. A New Implementation Technique for Applicative Languages. *Software—Practice and Experience*, 9(1):31–49, 1979.

32 C.P. Wadsworth. *Semantics and Pragmatics of the Lambda-calculus*. University of Oxford, 1971. URL: `https://books.google.pl/books?id=kl1QIQAACAAJ`.

Dependency Pairs Termination in Dependent Type Theory Modulo Rewriting

Frédéric Blanqui
INRIA, France
LSV, ENS Paris-Saclay, CNRS, Université Paris-Saclay, France

Guillaume Genestier
LSV, ENS Paris-Saclay, CNRS, Université Paris-Saclay, France
MINES ParisTech, PSL University, Paris, France

Olivier Hermant
MINES ParisTech, PSL University, Paris, France

Abstract

Dependency pairs are a key concept at the core of modern automated termination provers for first-order term rewriting systems. In this paper, we introduce an extension of this technique for a large class of dependently-typed higher-order rewriting systems. This extends previous results by Wahlstedt on the one hand and the first author on the other hand to strong normalization and non-orthogonal rewriting systems. This new criterion is implemented in the type-checker DEDUKTI.

2012 ACM Subject Classification Theory of computation → Equational logic and rewriting; Theory of computation → Type theory

Keywords and phrases termination, higher-order rewriting, dependent types, dependency pairs

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.9

Acknowledgements The authors thank the anonymous referees for their comments, which have improved the quality of this article.

1 Introduction

Termination, that is, the absence of infinite computations, is an important problem in software verification, as well as in logic. In logic, it is often used to prove cut elimination and consistency. In automated theorem provers and proof assistants, it is often used (together with confluence) to check decidability of equational theories and type-checking algorithms.

This paper introduces a new termination criterion for a large class of programs whose operational semantics can be described by higher-order rewriting rules [33] typable in the $\lambda\Pi$-calculus modulo rewriting ($\lambda\Pi/\mathcal{R}$ for short). $\lambda\Pi/\mathcal{R}$ is a system of dependent types where types are identified modulo the β-reduction of λ-calculus and a set $\mathcal{R}$ of rewriting rules given by the user to define not only functions but also types. It extends Barendregt's Pure Type System (PTS) λP [3], the logical framework LF [16] and Martin-Löf's type theory. It can encode any functional PTS like System F or the Calculus of Constructions [10].

Dependent types, introduced by de Bruijn in AUTOMATH, subsume generalized algebraic data types (GADT) used in some functional programming languages. They are at the core of many proof assistants and programming languages: COQ, TWELF, AGDA, LEAN, IDRIS, ...

Our criterion has been implemented in DEDUKTI, a type-checker for $\lambda\Pi/\mathcal{R}$ that we will use in our examples. The code is available in [12] and could be easily adapted to a subset of other languages like AGDA. As far as we know, this tool is the first one to automatically check termination in $\lambda\Pi/\mathcal{R}$, which includes both higher-order rewriting and dependent types.

This criterion is based on dependency pairs, an important concept in the termination of first-order term rewriting systems. It generalizes the notion of recursive call in first-

© Frédéric Blanqui, Guillaume Genestier, and Olivier Hermant;
licensed under Creative Commons License CC-BY
4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019).
Editor: Herman Geuvers; Article No. 9; pp. 9:1–9:21
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

order functional programs to rewriting. Namely, the dependency pairs of a rewriting rule $f(l_1,\dots,l_p) \to r$ are the pairs $(f(l_1,\dots,l_p), g(m_1,\dots,m_q))$ such that $g(m_1,\dots,m_q)$ is a subterm of r and g is a function symbol defined by some rewriting rules. Dependency pairs have been introduced by Arts and Giesl [2] and have evolved into a general framework for termination [13]. It is now at the heart of many state-of-the-art automated termination provers for first-order rewriting systems and Haskell, Java or C programs.

Dependency pairs have been extended to different simply-typed settings for higher-order rewriting: Combinatory Reduction Systems [23] and Higher-order Rewriting Systems [29], with two different approaches: dynamic dependency pairs include variable applications [24], while static dependency pairs exclude them by slightly restricting the class of systems that can be considered [25]. Here, we use the static approach.

In [37], Wahlstedt considered a system slightly less general than $\lambda\Pi/\mathcal{R}$ for which he provided conditions that imply the weak normalization, that is, the existence of a finite reduction to normal form. In his system, $\mathcal{R}$ uses matching on constructors only, like in the languages OCaml or Haskell. In this case, $\mathcal{R}$ is orthogonal: rules are left-linear (no variable occurs twice in a left-hand side) and have no critical pairs (no two rule left-hand side instances overlap). Wahlstedt's proof proceeds in two modular steps. First, he proves that typable terms have a normal form if there is no infinite sequence of function calls. Second, he proves that there is no infinite sequence of function calls if $\mathcal{R}$ satisfies Lee, Jones and Ben-Amram's size-change termination criterion (SCT) [26].

In this paper, we extend Wahlstedt's results in two directions. First, we prove a stronger normalization property: the absence of infinite reductions. Second, we assume that $\mathcal{R}$ is locally confluent, a much weaker condition than orthogonality: rules can be non-left-linear and have joinable critical pairs.

In [5], the first author developed a termination criterion for a calculus slightly more general than $\lambda\Pi/\mathcal{R}$, based on the notion of computability closure, assuming that type-level rules are orthogonal. The computability closure of a term $f(l_1,\dots,l_p)$ is a set of terms that terminate whenever $l_1,\dots,l_p$ terminate. It is defined inductively thanks to deduction rules preserving this property, using a precedence and a fixed well-founded ordering for dealing with function calls. Termination can then be enforced by requiring each rule right-hand side to belong to the computability closure of its corresponding left-hand side.

We extend this work as well by replacing that fixed ordering by the dependency pair relation. In [5], there must be a decrease in every function call. Using dependency pairs allows one to have non-strict decreases. Then, following Wahlstedt, SCT can be used to enforce the absence of infinite sequence of dependency pairs. But other criteria have been developed for this purpose that could be adapted to $\lambda\Pi/\mathcal{R}$.

Outline

The main result is Theorem 11 stating that, for a large class of rewriting systems $\mathcal{R}$, the combination of β and $\mathcal{R}$ is strongly normalizing on terms typable in $\lambda\Pi/\mathcal{R}$ if, roughly speaking, there is no infinite sequence of dependency pairs.

The proof involves two steps. First, after recalling the terms and types of $\lambda\Pi/\mathcal{R}$ in Section 2, we introduce in Section 3 a model of this calculus based on Girard's reducibility candidates [15], and prove that every typable term is strongly normalizing if every symbol of the signature is in the interpretation of its type (Adequacy lemma). Second, in Section 4, we introduce our notion of dependency pair and prove that every symbol of the signature is in the interpretation of its type if there is no infinite sequence of dependency pairs.

In order to show the usefulness of this result, we give simple criteria for checking the conditions of the theorem. In Section 5, we show that *plain function passing* systems belong to the class of systems that we consider. And in Section 6, we show how to use size-change termination to obtain the termination of the dependency pair relation.

Finally, in Section 7 we compare our criterion with other criteria and tools and, in Section 8, we summarize our results and give some hints on possible extensions.

For lack of space, some proofs are given in an appendix at the end of the paper.

2 Terms and types

The set $\mathbb{T}$ of terms of $\lambda\Pi/\mathcal{R}$ is the same as those of Barendregt's λP [3]:

$$t \in \mathbb{T} = s \in \mathbb{S} \mid x \in \mathbb{V} \mid f \in \mathbb{F} \mid \forall x : t, t \mid tt \mid \lambda x : t, t$$

where $\mathbb{S} = \{\text{TYPE}, \text{KIND}\}$ is the set of sorts[1], $\mathbb{V}$ is an infinite set of variables and $\mathbb{F}$ is a set of function symbols, so that $\mathbb{S}$, $\mathbb{V}$ and $\mathbb{F}$ are pairwise disjoint.

Furthermore, we assume given a set $\mathcal{R}$ of rules $l \to r$ such that $\text{FV}(r) \subseteq \text{FV}(l)$ and l is of the form $f\vec{l}$. A symbol f is said to be defined if there is a rule of the form $f\vec{l} \to r$. In this paper, we are interested in the termination of

$$\to \; = \; \to_\beta \cup \to_\mathcal{R}$$

where $\to_\beta$ is the β-reduction of λ-calculus and $\to_\mathcal{R}$ is the smallest relation containing $\mathcal{R}$ and closed by substitution and context: we consider rewriting with syntactic matching only. Following [6], it should however be possible to extend the present results to rewriting with matching modulo $\beta\eta$ or some equational theory. Let SN be the set of terminating terms and, given a term t, let $\to\!(t) = \{u \in \mathbb{T} \mid t \to u\}$ be the set of immediate reducts of t.

A typing environment Γ is a (possibly empty) sequence $x_1 : T_1, \ldots, x_n : T_n$ of pairs of variables and terms, where the variables are distinct, written $\vec{x} : \vec{T}$ for short. Given an environment $\Gamma = \vec{x} : \vec{T}$ and a term U, let $\forall\Gamma, U$ be $\forall\vec{x} : \vec{T}, U$. The product arity $\text{ar}(T)$ of a term T is the integer $n \in \mathbb{N}$ such that $T = \forall x_1 : T_1, \ldots \forall x_n : T_n, U$ and U is not a product. Let $\vec{t}$ denote a possibly empty sequence of terms $t_1, \ldots, t_n$ of length $|\vec{t}| = n$, and $\text{FV}(t)$ be the set of free variables of t.

For each $f \in \mathbb{F}$, we assume given a term Θ_f and a sort s_f, and let Γ_f be the environment such that $\Theta_f = \forall\Gamma_f, U$ and $|\Gamma_f| = \text{ar}(\Theta_f)$.

The application of a substitution σ to a term t is written $t\sigma$. Given a substitution σ, let $\text{dom}(\sigma) = \{x | x\sigma \neq x\}$, $\text{FV}(\sigma) = \bigcup_{x \in \text{dom}(\sigma)} \text{FV}(x\sigma)$ and $[x \mapsto a, \sigma]$ ($[x \mapsto a]$ if σ is the identity) be the substitution $\{(x, a)\} \cup \{(y, b) \in \sigma \mid y \neq x\}$. Given another substitution σ', let $\sigma \to \sigma'$ if there is x such that $x\sigma \to x\sigma'$ and, for all $y \neq x$, $y\sigma = y\sigma'$.

The typing rules of $\lambda\Pi/\mathcal{R}$, in Figure 1, add to those of λP the rule (fun) similar to (var). Moreover, (conv) uses $\downarrow$ instead of $\downarrow_\beta$, where $\downarrow \; = \; \to^* \, {}^*\!\!\leftarrow$ is the joinability relation and $\to^*$ the reflexive and transitive closure of $\to$. We say that t has type T in Γ if $\Gamma \vdash t : T$ is derivable. A substitution σ is well-typed from Δ to Γ, written $\Gamma \vdash \sigma : \Delta$, if, for all $(x : T) \in \Delta$, $\Gamma \vdash x\sigma : T\sigma$ holds.

The word "type" is used to denote a term occurring at the right-hand side of a colon in a typing judgment (and we usually use capital letters for types). Hence, KIND is the type of TYPE, Θ_f is the type of f, and s_f is the type of Θ_f. Common data types like natural numbers $\mathbb{N}$ are usually declared in $\lambda\Pi$ as function symbols of type TYPE: $\Theta_\mathbb{N} = \text{TYPE}$ and $s_\mathbb{N} = \text{KIND}$.

[1] Sorts refer here to the notion of sort in Pure Type Systems, not the one used in some first-order settings.

$$\text{(ax)}\quad \frac{}{\vdash \text{TYPE} : \text{KIND}} \qquad \text{(abs)}\quad \frac{\Gamma, x : A \vdash b : B \quad \Gamma \vdash (x : A)B : s}{\Gamma \vdash \lambda x : A.b : (x : A)B}$$

$$\text{(var)}\quad \frac{\Gamma \vdash A : s \quad x \notin \text{dom}(\Gamma)}{\Gamma, x : A \vdash x : A} \qquad \text{(app)}\quad \frac{\Gamma \vdash t : (x : A)B \quad \Gamma \vdash a : A}{\Gamma \vdash ta : B[x \mapsto a]}$$

$$\text{(weak)}\quad \frac{\Gamma \vdash A : s \quad \Gamma \vdash b : B \quad x \notin \text{dom}(\Gamma)}{\Gamma, x : A \vdash b : B} \qquad \text{(conv)}\quad \frac{\Gamma \vdash a : A \quad A \downarrow B \quad \Gamma \vdash B : s}{\Gamma \vdash a : B}$$

$$\text{(prod)}\quad \frac{\Gamma \vdash A : \text{TYPE} \quad \Gamma, x : A \vdash B : s}{\Gamma \vdash (x : A)B : s} \qquad \text{(fun)}\quad \frac{\vdash \Theta_f : s_f}{\vdash f : \Theta_f}$$

Figure 1 Typing rules of $\lambda\Pi/\mathcal{R}$.

The dependent product $\forall x : A, B$ generalizes the arrow type $A \Rightarrow B$ of simply-typed λ-calculus: it is the type of functions taking an argument x of type A and returning a term whose type B may depend on x. If B does not depend on x, we sometimes simply write $A \Rightarrow B$.

Typing induces a hierarchy on terms [4, Lemma 47]. At the top, there is the sort KIND that is not typable. Then, comes the class $\mathbb{K}$ of kinds, whose type is KIND: $K = \text{TYPE} \mid \forall x : t, K$ where $t \in \mathbb{T}$. Then, comes the class of predicates, whose types are kinds. Finally, at the bottom lie (proof) objects whose types are predicates.

▶ **Example 1** (Filter function on dependent lists)**.** To illustrate the kind of systems we consider, we give an extensive example in the new DEDUKTI syntax combining type-level rewriting rules (`El` converts datatype codes into DEDUKTI types), dependent types ($\mathbb{L}$ is the polymorphic type of lists parameterized with their length), higher-order variables (`fil` is a function filtering elements out of a list along a boolean function `f`), and matching on defined function symbols (`fil` can match a list defined by concatenation). Note that this example cannot be represented in COQ or AGDA because of the rules using matching on `app`. And its termination can be handled neither by [37] nor by [5] because the system is not orthogonal and has no strict decrease in every recursive call. It can however be handled by our new termination criterion and its implementation [12]. For readability, we removed the `&` which are used to identify pattern variables in the rewriting rules.

```
symbol Set: TYPE      symbol arrow: Set ⇒ Set ⇒ Set

symbol El: Set ⇒ TYPE         rule El (arrow a b) → El a ⇒ El b

symbol Bool: TYPE       symbol true: Bool      symbol false: Bool
symbol Nat: TYPE        symbol zero: Nat       symbol s: Nat ⇒ Nat

symbol plus: Nat ⇒ Nat ⇒Nat      set infix 1 "+" ≔ plus
  rule zero + q → q               rule (s p) + q → s (p + q)

symbol List: Set ⇒ Nat ⇒ TYPE
  symbol nil: ∀a, List a zero
  symbol cons:∀a, El a ⇒ ∀p, List a p ⇒ List a (s p)

symbol app: ∀a p, List a p ⇒ ∀q, List a q ⇒ List a (p+q)
  rule app a _ (nil _)          q m → m
  rule app a _ (cons _ x p l) q m → cons a x (p+q) (app a p l q m)

symbol len_fil: ∀a, (El a ⇒ Bool) ⇒ ∀p, List a p ⇒ Nat
symbol len_fil_aux: Bool ⇒ ∀a, (El a ⇒ Bool) ⇒ ∀p, List a p ⇒ Nat
```

```
  rule len_fil a f _ (nil _)           → zero
  rule len_fil a f _ (cons _ x p l)  → len_fil_aux (f x) a f p l
  rule len_fil a f _ (app _ p l q m)
       → (len_fil a f p l) + (len_fil a f q m)
  rule len_fil_aux true  a f p l → s (len_fil a f p l)
  rule len_fil_aux false a f p l → len_fil a f p l

symbol fil: ∀a f p l, List a (len_fil a f p l)
symbol fil_aux: ∀b a f, El a ⇒ ∀p l, List a (len_fil_aux b a f p l)
  rule fil a f _ (nil _)          → nil a
  rule fil a f _ (cons _ x p l)  → fil_aux (f x) a f x p l
  rule fil a f _ (app _ p l q m)
       → app a (len_fil a f p l) (fil a f p l)
               (len_fil a f q m) (fil a f q m)
  rule fil_aux false a f x p l → fil a f p l
  rule fil_aux true  a f x p l
       → cons a x (len_fil a f p l) (fil a f p l)
```

Assumptions. Throughout the paper, we assume that $\to$ is locally confluent ($\leftarrow\!\!\to \subseteq \downarrow$) and preserves typing (for all Γ, A, t and u, if $\Gamma \vdash t : A$ and $t \to u$, then $\Gamma \vdash u : A$).

Note that local confluence implies that every $t \in \mathrm{SN}$ has a unique normal form $t\!\downarrow$.

These assumptions are used in the interpretation of types (Definition 2) and the adequacy lemma (Lemma 5). Both properties are undecidable in general. For confluence, Dedukti can call confluence checkers that understand the HRS format of the confluence competition. For preservation of typing by reduction, it implements an heuristic [31].

3 Interpretation of types as reducibility candidates

We aim to prove the termination of the union of two relations, $\to_\beta$ and $\to_\mathcal{R}$, on the set of well-typed terms (which depends on $\mathcal{R}$ since $\downarrow$ includes $\to_\mathcal{R}$). As is well known, termination is not modular in general. As a β step can generate an $\mathcal{R}$ step, and vice versa, we cannot expect to prove the termination of $\to_\beta \cup \to_\mathcal{R}$ from the termination of $\to_\beta$ and $\to_\mathcal{R}$. The termination of $\lambda\Pi/\mathcal{R}$ cannot be reduced to the termination of the simply-typed λ-calculus either (as done for $\lambda\Pi$ alone in [16]) because of type-level rewriting rules like the ones defining `El` in Example 1. Indeed, type-level rules enable the encoding of functional PTS like Girard's System F, whose termination cannot be reduced to the termination of the simply-typed λ-calculus [10].

So, following Girard [15], to prove the termination of $\to_\beta \cup \to_\mathcal{R}$, we build a model of our calculus by interpreting types into sets of terminating terms. To this end, we need to find an interpretation $[\![\ ]\!]$ having the following properties:

- Because types are identified modulo conversion, we need $[\![\ ]\!]$ to be invariant by reduction: if T is typable and $T \to T'$, then we must have $[\![T]\!] = [\![T']\!]$.
- As usual, to handle β-reduction, we need a product type $\forall x : A, B$ to be interpreted by the set of terms t such that, for all a in the interpretation of A, ta is in the interpretation of $B[x \mapsto a]$, that is, we must have $[\![\forall x : A, B]\!] = \Pi a \in [\![A]\!].\, [\![B[x \mapsto a]]\!]$ where $\Pi a \in P.\, Q(a) = \{t \mid \forall a \in P, ta \in Q(a)\}$.

First, we define the interpretation of predicates (and TYPE) as the least fixpoint of a monotone function in a directed-complete (= chain-complete) partial order [28]. Second, we define the interpretation of kinds by induction on their size.

▶ **Definition 2** (Interpretation of types). *Let $\mathbb{I} = \mathcal{F}_p(\mathbb{T}, \mathcal{P}(\mathbb{T}))$ be the set of partial functions from $\mathbb{T}$ to the powerset of $\mathbb{T}$. It is directed-complete wrt inclusion, allowing us to define $\mathcal{I}$ as the least fixpoint of the monotone function $F : \mathbb{I} \to \mathbb{I}$ such that, if $I \in \mathbb{I}$, then:*

- *The domain of $F(I)$ is the set $D(I)$ of all the terminating terms T such that, if T reduces to some product term $\forall x : A, B$ (not necessarily in normal form), then $A \in \mathrm{dom}(I)$ and, for all $a \in I(A)$, $B[x \mapsto a] \in \mathrm{dom}(I)$.*
- *If $T \in D(I)$ and the normal form[2] of T is not a product, then $F(I)(T) = \mathrm{SN}$.*
- *If $T \in D(I)$ and $T{\downarrow} = \forall x : A, B$, then $F(I)(T) = \Pi a \in I(A).\, I(B[x \mapsto a])$.*

We now introduce $\mathcal{D} = D(\mathcal{I})$ and define the interpretation of a term T wrt to a substitution σ, $[\![T]\!]_\sigma$ (and simply $[\![T]\!]$ if σ is the identity), as follows:

- *$[\![s]\!]_\sigma = \mathcal{D}$ if $s \in \mathbb{S}$,*
- *$[\![\forall x : A, K]\!]_\sigma = \Pi a \in [\![A]\!]_\sigma.\, [\![K]\!]_{[x \mapsto a, \sigma]}$ if $K \in \mathbb{K}$ and $x \notin \mathrm{dom}(\sigma)$,*
- *$[\![T]\!]_\sigma = \mathcal{I}(T\sigma)$ if $T \notin \mathbb{K} \cup \{\mathrm{KIND}\}$ and $T\sigma \in \mathcal{D}$,*
- *$[\![T]\!]_\sigma = \mathrm{SN}$ otherwise.*

A substitution σ is adequate wrt an environment Γ, $\sigma \models \Gamma$, if, for all $x : A \in \Gamma$, $x\sigma \in [\![A]\!]_\sigma$. A typing map Θ is adequate if, for all f, $f \in [\![\Theta_f]\!]$ whenever $\vdash \Theta_f : s_f$ and $\Theta_f \in [\![s_f]\!]$.

Let $\mathbb{C}$ be the set of terms of the form $f\vec{t}$ such that $|\vec{t}| = \mathrm{ar}(\Theta_f)$, $\vdash \Theta_f : s_f$, $\Theta_f \in [\![s_f]\!]$ and, if $\Gamma_f = \vec{x} : \vec{A}$ and $\sigma = [\vec{x} \mapsto \vec{t}]$, then $\sigma \models \Gamma_f$. (Informally, $\mathbb{C}$ is the set of terms obtained by fully applying some function symbol to computable arguments.)

We can then prove that, for all terms T, $[\![T]\!]$ satisfies Girard's conditions of reducibility candidates, called computability predicates here, adapted to rewriting by including in neutral terms every term of the form $f\vec{t}$ when f is applied to enough arguments wrt $\mathcal{R}$ [5]:

▶ **Definition 3** (Computability predicates). *A term is neutral if it is of the form $(\lambda x : A, t)u\vec{v}$, $x\vec{v}$ or $f\vec{v}$ with, for every rule $f\vec{l} \to r \in \mathcal{R}$, $|\vec{l}| \leq |\vec{v}|$.*

Let $\mathbb{P}$ be the set of all the sets of terms S (computability predicates) such that (a) $S \subseteq \mathrm{SN}$, (b) $\to(S) \subseteq S$, and (c) $t \in S$ if t is neutral and $\to(t) \subseteq S$.

Note that neutral terms satisfy the following key property: if t is neutral then, for all u, tu is neutral and every reduct of tu is either of the form $t'u$ with t' a reduct of t, or of the form tu' with u' a reduct of u.

One can easily check that SN is a computability predicate.

Note also that a computability predicate is never empty: it contains every neutral term in normal form. In particular, it contains every variable.

We then get the following results (the proofs are given in Appendix A):

▶ **Lemma 4.**

(a) *For all terms T and substitutions σ, $[\![T]\!]_\sigma \in \mathbb{P}$.*
(b) *If T is typable, $T\sigma \in \mathcal{D}$ and $T \to T'$, then $[\![T]\!]_\sigma = [\![T']\!]_\sigma$.*
(c) *If T is typable, $T\sigma \in \mathcal{D}$ and $\sigma \to \sigma'$, then $[\![T]\!]_\sigma = [\![T]\!]_{\sigma'}$.*
(d) *If $\forall x : A, B$ is typable and $\forall x : A\sigma, B\sigma \in \mathcal{D}$, then $[\![\forall x : A, B]\!]_\sigma = \Pi a \in [\![A]\!]_\sigma.\, [\![B]\!]_{[x \mapsto a, \sigma]}$.*
(e) *If $\Delta \vdash U : s$, $\Gamma \vdash \gamma : \Delta$ and $U\gamma\sigma \in \mathcal{D}$, then $[\![U\gamma]\!]_\sigma = [\![U]\!]_{\gamma\sigma}$.*
(f) *Given $P \in \mathbb{P}$ and, for all $a \in P$, $Q(a) \in \mathbb{P}$ such that $Q(a') \subseteq Q(a)$ if $a \to a'$. Then, $\lambda x : A, b \in \Pi a \in P.\, Q(a)$ if $A \in \mathrm{SN}$ and, for all $a \in P$, $b[x \mapsto a] \in Q(a)$.*

We can finally prove that our model is adequate, that is, every term of type T belongs to $[\![T]\!]$, if the typing map Θ itself is adequate. This reduces the termination of well-typed terms to the computability of function symbols.

[2] Because we assume local confluence, every terminating term T has a unique normal form $T{\downarrow}$.

▶ **Lemma 5** (Adequacy). *If Θ is adequate, $\Gamma \vdash t : T$ and $\sigma \models \Gamma$, then $t\sigma \in [\![T]\!]_\sigma$.*

Proof. First note that, if $\Gamma \vdash t : T$, then either $T = \textsc{kind}$ or $\Gamma \vdash T : s$ [4, Lemma 28]. Moreover, if $\Gamma \vdash a : A$, $A \downarrow B$ and $\Gamma \vdash B : s$ (the premises of the (conv) rule), then $\Gamma \vdash A : s$ [4, Lemma 42] (because $\to$ preserves typing). Hence, the relation $\vdash$ is unchanged if one adds the premise $\Gamma \vdash A : s$ in (conv), giving the rule (conv'). Similarly, we add the premise $\Gamma \vdash \forall x : A, B : s$ in (app), giving the rule (app'). We now prove the lemma by induction on $\Gamma \vdash t : T$ using (app') and (conv'):

(ax) It is immediate that $\textsc{type} \in [\![\textsc{kind}]\!]_\sigma = \mathcal{D}$.

(var) By assumption on σ.

(weak) If $\sigma \models \Gamma, x : A$, then $\sigma \models \Gamma$. So, the result follows by induction hypothesis.

(prod) Is $(\forall x : A, B)\sigma$ in $[\![s]\!]_\sigma = \mathcal{D}$? Wlog we can assume $x \notin \mathrm{dom}(\sigma) \cup \mathrm{FV}(\sigma)$. So, $(\forall x : A, B)\sigma = \forall x : A\sigma, B\sigma$. By induction hypothesis, $A\sigma \in [\![\textsc{type}]\!]_\sigma = \mathcal{D}$. Let now $a \in \mathcal{I}(A\sigma)$ and $\sigma' = [x \mapsto a, \sigma]$. Note that $\mathcal{I}(A\sigma) = [\![A]\!]_\sigma$. So, $\sigma' \models \Gamma, x : A$ and, by induction hypothesis, $B\sigma' \in [\![s]\!]_\sigma = \mathcal{D}$. Since $x \notin \mathrm{dom}(\sigma) \cup \mathrm{FV}(\sigma)$, we have $B\sigma' = (B\sigma)[x \mapsto a]$. Therefore, $(\forall x : A, B)\sigma \in [\![s]\!]_\sigma$.

(abs) Is $(\lambda x : A, b)\sigma$ in $[\![\forall x : A, B]\!]_\sigma$? Wlog we can assume that $x \notin \mathrm{dom}(\sigma) \cup \mathrm{FV}(\sigma)$. So, $(\lambda x : A, b)\sigma = \lambda x : A\sigma, b\sigma$. By Lemma 4d, $[\![\forall x : A, B]\!]_\sigma = \Pi a \in [\![A]\!]_\sigma. [\![B]\!]_{[x \mapsto a, \sigma]}$. By Lemma 4c, $[\![B]\!]_{[x \mapsto a, \sigma]}$ is an $[\![A]\!]_\sigma$-indexed family of computability predicates such that $[\![B]\!]_{[x \mapsto a', \sigma]} = [\![B]\!]_{[x \mapsto a, \sigma]}$ whenever $a \to a'$. Hence, by Lemma 4f, $\lambda x : A\sigma, b\sigma \in [\![\forall x : A, B]\!]_\sigma$ if $A\sigma \in \mathrm{SN}$ and, for all $a \in [\![A]\!]_\sigma$, $(b\sigma)[x \mapsto a] \in [\![B]\!]_{\sigma'}$ where $\sigma' = [x \mapsto a, \sigma]$. By induction hypothesis, $(\forall x : A, B)\sigma \in [\![s]\!]_\sigma = \mathcal{D}$. Since $x \notin \mathrm{dom}(\sigma) \cup \mathrm{FV}(\sigma)$, $(\forall x : A, B)\sigma = \forall x : A\sigma, B\sigma$ and $(b\sigma)[x \mapsto a] = b\sigma'$. Since $\mathcal{D} \subseteq \mathrm{SN}$, we have $A\sigma \in \mathrm{SN}$. Moreover, since $\sigma' \models \Gamma, x : A$, we have $b\sigma' \in [\![B]\!]_{\sigma'}$ by induction hypothesis.

(app') Is $(ta)\sigma = (t\sigma)(a\sigma)$ in $[\![B[x \mapsto a]]\!]_\sigma$? By induction hypothesis, $t\sigma \in [\![\forall x : A, B]\!]_\sigma$, $a\sigma \in [\![A]\!]_\sigma$ and $(\forall x : A, B)\sigma \in [\![s]\!] = \mathcal{D}$. By Lemma 4d, $[\![\forall x : A, B]\!]_\sigma = \Pi\alpha \in [\![A]\!]_\sigma. [\![B]\!]_{[x \mapsto \alpha, \sigma]}$. Hence, $(t\sigma)(a\sigma) \in [\![B]\!]_{\sigma'}$ where $\sigma' = [x \mapsto a\sigma, \sigma]$. Wlog we can assume $x \notin \mathrm{dom}(\sigma) \cup \mathrm{FV}(\sigma)$. So, $\sigma' = [x \mapsto a]\sigma$. Hence, by Lemma 4e, $[\![B]\!]_{\sigma'} = [\![B[x \mapsto a]]\!]_\sigma$.

(conv') By induction hypothesis, $a\sigma \in [\![A]\!]_\sigma$, $A\sigma \in [\![s]\!]_\sigma = \mathcal{D}$ and $B\sigma \in [\![s]\!]_\sigma = \mathcal{D}$. By Lemma 4b, $[\![A]\!]_\sigma = [\![B]\!]_\sigma$. So, $a\sigma \in [\![B]\!]_\sigma$.

(fun) By induction hypothesis, $\Theta_f \in [\![s_f]\!]_\sigma = \mathcal{D}$. Therefore, $f \in [\![\Theta_f]\!]_\sigma = [\![\Theta_f]\!]$ since Θ is adequate. ◀

4 Dependency pairs theorem

Now, we prove that the adequacy of Θ can be reduced to the absence of infinite sequences of dependency pairs, as shown by Arts and Giesl for first-order rewriting [2].

▶ **Definition 6** (Dependency pairs). *Let $f\vec{l} > g\vec{m}$ iff there is a rule $f\vec{l} \to r \in \mathcal{R}$, g is defined and $g\vec{m}$ is a subterm of r such that $\vec{m}$ are all the arguments to which g is applied. The relation $>$ is the set of dependency pairs.*

Let $\tilde{>} = \to_{\mathrm{arg}}^* >_s$ be the relation on the set $\mathbb{C}$ (Def. 2), where $f\vec{t} \to_{\mathrm{arg}} f\vec{u}$ iff $\vec{t} \to_{prod} \vec{u}$ (reduction in one argument), and $>_s$ is the closure by substitution and left-application of $>$: $ft_1 \dots t_p \,\tilde{>}\, gu_1 \dots u_q$ iff there are a dependency pair $fl_1 \dots l_i > gm_1 \dots m_j$ with $i \leq p$ and $j \leq q$ and a substitution σ such that, for all $k \leq i$, $t_k \to^* l_k\sigma$ and, for all $k \leq j$, $m_k\sigma = u_k$.

In our setting, we have to close $>_s$ by left-application because function symbols are curried. When a function symbol f is not fully applied wrt $\mathrm{ar}(\Theta_f)$, the missing arguments must be considered as potentially being anything. Indeed, the following rewriting system:

```
      app x y → x y                f x y → app (f x) y
```

whose dependency pairs are `f x y > app (f x) y` and `f x y > f x`, does not terminate, but there is no way to construct an infinite sequence of dependency pairs without adding an argument to the right-hand side of the second dependency pair.

▶ **Example 7.** The rules of Example 1 have the following dependency pairs (the pairs whose left-hand side is headed by `fil` or `fil_aux` can be found in Appendix B):

```
A:                El (arrow a b) > El a
B:                El (arrow a b) > El b
C:                     (s p) + q > p + q
D:   app a _ (cons _ x p l) q m > p + q
E:   app a _ (cons _ x p l) q m > app a p l q m
F:len_fil a f _ (cons _ x p l)  > len_fil_aux (f x) a f p l
G:len_fil a f _ (app _ p l q m) >
                                 (len_fil a f p l) + (len_fil a f q m)
H:len_fil a f _ (app _ p l q m) > len_fil a f p l
I:len_fil a f _ (app _ p l q m) > len_fil a f q m
J:    len_fil_aux true  a f p l > len_fil a f p l
K:    len_fil_aux false a f p l > len_fil a f p l
```

In [2], a sequence of dependency pairs interleaved with $\to_{\text{arg}}$ steps is called a chain. Arts and Giesl proved that, in a first-order term algebra, $\to_{\mathcal{R}}$ terminates if and only if there are no infinite chains, that is, if and only if $\tilde{>}$ terminates. Moreover, in a first-order term algebra, $\tilde{>}$ terminates if and only if, for all f and $\vec{t}$, $f\vec{t}$ terminates wrt $\tilde{>}$ whenever $\vec{t}$ terminates wrt $\to$. In our framework, this last condition is similar to saying that Θ is adequate.

We now introduce the class of systems to which we will extend Arts and Giesl's theorem.

▶ **Definition 8** (Well-structured system). *Let $\succeq$ be the smallest quasi-order on $\mathbb{F}$ such that $f \succeq g$ if g occurs in Θ_f or if there is a rule $f\vec{l} \to r \in \mathcal{R}$ with g (defined or undefined) occurring in r. Then, let $\succ\ =\ \succeq \setminus \preceq$ be the strict part of $\succeq$. A rewriting system $\mathcal{R}$ is well-structured if:*

(a) *$\succ$ is well-founded;*

(b) *for every rule $f\vec{l} \to r$, $|\vec{l}| \leq \mathrm{ar}(\Theta_f)$;*

(c) *for every dependency pair $f\vec{l} > g\vec{m}$, $|\vec{m}| \leq \mathrm{ar}(\Theta_g)$;*

(d) *every rule $f\vec{l} \to r$ is equipped with an environment $\Delta_{f\vec{l}\to r}$ such that, if $\Theta_f = \forall\vec{x} : \vec{T}, U$ and $\pi = [\vec{x} \mapsto \vec{l}]$, then $\Delta_{f\vec{l}\to r} \vdash_{f\vec{l}} r : U\pi$, where $\vdash_{f\vec{l}}$ is the restriction of $\vdash$ defined in Fig. 2.*

Condition (a) is always satisfied when $\mathbb{F}$ is finite. Condition (b) ensures that a term of the form $f\vec{t}$ is neutral whenever $|\vec{t}| = \mathrm{ar}(\Theta_f)$. Condition (c) ensures that $>$ is included in $\tilde{>}$.

The relation $\vdash_{f\vec{l}}$ corresponds to the notion of computability closure in [5], with the ordering on function calls replaced by the dependency pair relation. It is similar to $\vdash$ except that it uses the variant of (conv) and (app) used in the proof of the adequacy lemma; (fun) is split in the rules (const) for undefined symbols and (dp) for dependency pairs whose left-hand side is $f\vec{l}$; every type occurring in an object term or every type of a function symbol occurring in a term is required to be typable by using symbols smaller than f only.

The environment $\Delta_{f\vec{l}\to r}$ can be inferred by Dedukti when one restricts rule left hand-sides to some well-behaved class of terms like algebraic terms or Miller patterns (in λProlog).

One can check that Example 1 is well-structured (the proof is given in Appendix B).

Finally, we need matching to be compatible with computability, that is, if $f\vec{l} \to r \in \mathcal{R}$ and $\vec{l}\sigma$ are computable, then σ is computable, a condition called accessibility in [5]:

$$\text{(ax)} \quad \frac{}{\vdash_{f\vec{l}} \text{TYPE} : \text{KIND}} \qquad \text{(weak)} \quad \frac{\Gamma \vdash_{\prec f} A : s \quad \Gamma \vdash_{f\vec{l}} b : B \quad x \notin \text{dom}(\Gamma)}{\Gamma, x : A \vdash_{f\vec{l}} b : B}$$

$$\text{(var)} \quad \frac{\Gamma \vdash_{\prec f} A : s \quad x \notin \text{dom}(\Gamma)}{\Gamma, x : A \vdash_{f\vec{l}} x : A} \qquad \text{(prod)} \quad \frac{\Gamma \vdash_{f\vec{l}} A : \text{TYPE} \quad \Gamma, x : A \vdash_{f\vec{l}} B : s}{\Gamma \vdash_{f\vec{l}} \forall x : A, B : s}$$

$$\text{(abs)} \quad \frac{\Gamma, x : A \vdash_{f\vec{l}} b : B \quad \Gamma \vdash_{\prec f} \forall x : A, B : s}{\Gamma \vdash_{f\vec{l}} \lambda x : A, b : \forall x : A, B}$$

$$\text{(app')} \quad \frac{\Gamma \vdash_{f\vec{l}} t : \forall x : A, B \quad \Gamma \vdash_{f\vec{l}} a : A \quad \Gamma \vdash_{\prec f} \forall x : A, B : s}{\Gamma \vdash_{f\vec{l}} ta : B[x \mapsto a]}$$

$$\text{(conv')} \quad \frac{\Gamma \vdash_{f\vec{l}} a : A \quad A \downarrow B \quad \Gamma \vdash_{\prec f} B : s \quad \Gamma \vdash_{\prec f} A : s}{\Gamma \vdash_{f\vec{l}} a : B}$$

$$\text{(dp)} \quad \frac{\vdash_{\prec f} \Theta_g : s_g \quad \Gamma \vdash_{f\vec{l}} \gamma : \Sigma}{\Gamma \vdash_{f\vec{l}} g\vec{y}\gamma : V\gamma} \quad (\Theta_g = (\forall \vec{y} : \vec{U}, V), \Sigma = \vec{y} : \vec{U}, g\vec{y}\gamma < f\vec{l})$$

$$\text{(const)} \quad \frac{\vdash_{\prec f} \Theta_g : s_g}{\vdash_{f\vec{l}} g : \Theta_g} \quad (g \text{ undefined})$$

and $\vdash_{\prec f}$ is defined by the same rules as $\vdash$, except (fun) replaced by:

$$\text{(fun}_{\prec f}\text{)} \quad \frac{\vdash_{\prec f} \Theta_g : s_g \quad g \prec f}{\vdash_{\prec f} g : \Theta_g}$$

Figure 2 Restricted type systems $\vdash_{f\vec{l}}$ and $\vdash_{\prec f}$.

▶ **Definition 9** (Accessible system). *A well-structured system $\mathcal{R}$ is accessible if, for all substitutions σ and rules $f\vec{l} \to r$ with $\Theta_f = \forall \vec{x} : \vec{T}, U$ and $|\vec{x}| = |\vec{l}|$, we have $\sigma \models \Delta_{f\vec{l}\to r}$ whenever $\vdash \Theta_f : s_f$, $\Theta_f \in [\![s_f]\!]$ and $[\vec{x} \mapsto \vec{l}]\sigma \models \vec{x} : \vec{T}$.*

This property is not always satisfied because the subterm relation does not preserve computability in general. Indeed, if C is an undefined type constant, then $[\![C]\!] = \text{SN}$. However, $[\![C \Rightarrow C]\!] \neq \text{SN}$ since $\omega = \lambda x : C, xx \in \text{SN}$ and $\omega\omega \notin \text{SN}$. Hence, if c is an undefined function symbol of type $\Theta_c = (C \Rightarrow C) \Rightarrow C$, then $c\omega \in [\![C]\!]$ but $\omega \notin [\![C \Rightarrow C]\!]$.

We can now state the main lemma:

▶ **Lemma 10.** *Θ is adequate if $\tilde{>}$ terminates and $\mathcal{R}$ is well-structured and accessible.*

Proof. Since $\mathcal{R}$ is well-structured, $\succ$ is well-founded by condition (a). We prove that, for all $f \in \mathbb{F}$, $f \in [\![\Theta_f]\!]$, by induction on $\succ$. So, let $f \in \mathbb{F}$ with $\Theta_f = \forall \Gamma_f, U$ and $\Gamma_f = x_1 : T_1, \ldots, x_n : T_n$. By induction hypothesis, we have that, for all $g \prec f$, $g \in [\![\Theta_g]\!]$.

Since $\to_{\text{arg}}$ and $\tilde{>}$ terminate on $\mathbb{C}$ and $\to_{\text{arg}} \tilde{>} \subseteq \tilde{>}$, we have that $\to_{\text{arg}} \cup \tilde{>}$ terminates. We now prove that, for all $f\vec{t} \in \mathbb{C}$, we have $f\vec{t} \in [\![U]\!]_\theta$ where $\theta = [\vec{x} \mapsto \vec{t}]$, by a second induction on $\to_{\text{arg}} \cup \tilde{>}$. By condition (b), $f\vec{t}$ is neutral. Hence, by definition of computability, it suffices to prove that, for all $u \in {\to}(f\vec{t})$, $u \in [\![U]\!]_\theta$. There are 2 cases:

- $u = f\vec{v}$ with $\vec{t} \to_{prod} \vec{v}$. Then, we can conclude by the first induction hypothesis.
- There are $fl_1 \ldots l_k \to r \in \mathcal{R}$ and σ such that $u = (r\sigma)t_{k+1} \ldots t_n$ and, for all $i \in \{1, \ldots, k\}$, $t_i = l_i\sigma$. Since $f\vec{t} \in \mathbb{C}$, we have $\pi\sigma \models \Gamma_f$. Since $\mathcal{R}$ is accessible, we get that $\sigma \models \Delta_{f\vec{l}\to r}$. By condition (d), we have $\Delta_{f\vec{l}\to r} \vdash_{f\vec{l}} r : V\pi$ where $V = \forall x_{k+1} : T_{k+1}, \ldots \forall x_n : T_n, U$. Now, we prove that, for all Γ, t and T, if $\Gamma \vdash_{f\vec{l}} t : T$ ($\Gamma \vdash_{\prec f} t : T$ resp.) and $\sigma \models \Gamma$, then $t\sigma \in [\![T]\!]_\sigma$, by a third induction on the structure of the derivation of $\Gamma \vdash_{f\vec{l}} t : T$ ($\Gamma \vdash_{\prec f} t : T$ resp.), as in the proof of Lemma 5 except for (fun) replaced by (fun$_{\prec f}$) in one case, and (const) and (dp) in the other case.
  (fun$_{\prec f}$) We have $g \in [\![\Theta_g]\!]$ by the first induction hypothesis on g.

(const) Since g is undefined, it is neutral and normal. Therefore, it belongs to every computability predicate and in particular to $[\![\Theta_g]\!]_\sigma$.

(dp) By the third induction hypothesis, $y_i\gamma\sigma \in [\![U_i\gamma]\!]_\sigma$. By Lemma 4e, $[\![U_i\gamma]\!]_\sigma = [\![U_i]\!]_{\gamma\sigma}$. So, $\gamma\sigma \models \Sigma$ and $g\vec{y}\gamma\sigma \in \mathbb{C}$. Now, by condition (c), $g\vec{y}\gamma\sigma \tilde{<} f\vec{l}\sigma$ since $g\vec{y}\gamma < f\vec{l}$. Therefore, by the second induction hypothesis, $g\vec{y}\gamma\sigma \in [\![V\gamma]\!]_\sigma$.

So, $r\sigma \in [\![V\pi]\!]_\sigma$ and, by Lemma 4d, $u \in [\![U]\!]_{[x_n \mapsto t_n,..,x_{k+1} \mapsto t_{k+1}, \pi\sigma]} = [\![U]\!]_\theta$. ◀

Note that the proof still works if one replaces the relation $\succeq$ of Definition 8 by any well-founded quasi-order such that $f \succeq g$ whenever $f\vec{l} > g\vec{m}$. The quasi-order of Definition 8, defined syntactically, relieves the user of the burden of providing one and is sufficient in every practical case met by the authors. However it is possible to construct ad-hoc systems which require a quasi-order richer than the one presented here.

By combining the previous lemma and the Adequacy lemma (the identity substitution is computable), we get the main result of the paper:

▶ **Theorem 11.** *The relation $\to = \to_\beta \cup \to_\mathcal{R}$ terminates on terms typable in $\lambda\Pi/\mathcal{R}$ if $\to$ is locally confluent and preserves typing, $\mathcal{R}$ is well-structured and accessible, and $\tilde{>}$ terminates.*

For the sake of completeness, we are now going to give sufficient conditions for accessibility and termination of $\tilde{>}$ to hold, but one could imagine many other criteria.

5 Checking accessibility

In this section, we give a simple condition to ensure accessibility and some hints on how to modify the interpretation when this condition is not satisfied.

As seen with the definition of accessibility, the main problem is to deal with subterms of higher-order type. A simple condition is to require higher-order variables to be direct subterms of the left-hand side, a condition called plain function-passing (PFP) in [25], and satisfied by Example 1.

▶ **Definition 12** (PFP systems)**.** *A well-structured $\mathcal{R}$ is PFP if, for all $f\vec{l} \to r \in \mathcal{R}$ with $\Theta_f = \forall \vec{x} : \vec{T}, U$ and $|\vec{x}| = |\vec{l}|$, $\vec{l} \notin \mathbb{K} \cup \{\text{KIND}\}$ and, for all $y : T \in \Delta_{f\vec{l} \to r}$, there is i such that $y = l_i$ and $T = T_i[\vec{x} \mapsto \vec{l}]$, or else $y \in \mathrm{FV}(l_i)$ and $T = D\vec{t}$ with D undefined and $|\vec{t}| = \mathrm{ar}(D)$.*

▶ **Lemma 13.** *PFP systems are accessible.*

Proof. Let $f\vec{l} \to r$ be a PFP rule with $\Theta_f = \forall\Gamma, U$, $\Gamma = \vec{x} : \vec{T}$, $\pi = [\vec{x} \mapsto \vec{l}]$. Following Definition 9, assume that $\vdash \Theta_f : s_f$, $\Theta_f \in \mathcal{D}$ and $\pi\sigma \models \Gamma$. We have to prove that, for all $(y : T) \in \Delta_{f\vec{l} \to r}$, $y\sigma \in [\![T]\!]_\sigma$.

- Suppose $y = l_i$ and $T = T_i\pi$. Then, $y\sigma = l_i\sigma \in [\![T_i]\!]_{\pi\sigma}$. Since $\vdash \Theta_f : s_f$, $T_i \notin \mathbb{K} \cup \{\text{KIND}\}$. Since $\Theta_f \in \mathcal{D}$ and $\pi\sigma \models \Gamma$, we have $T_i\pi\sigma \in \mathcal{D}$. So, $[\![T_i]\!]_{\pi\sigma} = \mathcal{I}(T_i\pi\sigma)$. Since $T_i \notin \mathbb{K} \cup \{\text{KIND}\}$ and $\vec{l} \notin \mathbb{K} \cup \{\text{KIND}\}$, $T_i\pi \notin \mathbb{K} \cup \{\text{KIND}\}$. Since $T_i\pi\sigma \in \mathcal{D}$, $[\![T_i\pi]\!]_\sigma = \mathcal{I}(T_i\pi\sigma)$. Thus, $y\sigma \in [\![T]\!]_\sigma$.
- Suppose $y \in \mathrm{FV}(l_i)$ and T is of the form $C\vec{t}$ with $|\vec{t}| = \mathrm{ar}(C)$. Then, $[\![T]\!]_\sigma = \mathrm{SN}$ and $y\sigma \in \mathrm{SN}$ since $l_i\sigma \in [\![T_i]\!]_\sigma \subseteq \mathrm{SN}$. ◀

But many accessible systems are not PFP. They can be proved accessible by changing the interpretation of type constants (a complete development is left for future work).

▶ **Example 14** (Recursor on Brouwer ordinals).

```
symbol Ord: TYPE
 symbol zero: Ord   symbol suc: Ord⇒Ord    symbol lim: (Nat⇒Ord)⇒Ord

symbol ordrec: A⇒(Ord⇒A⇒A)⇒((Nat⇒Ord)⇒(Nat⇒A)⇒A)⇒Ord⇒A
  rule ordrec u v w zero    → u
  rule ordrec u v w (suc x) → v x (ordrec u v w x)
  rule ordrec u v w (lim f) → w f (λn,ordrec u v w (f n))
```

The above example is not PFP because `f:Nat⇒Ord` is not argument of `ordrec`. Yet, it is accessible if one takes for $[\![\texttt{Ord}]\!]$ the least fixpoint of the monotone function $F(S) = \{t \in \mathrm{SN} \mid \text{if } t \to^* \texttt{lim}\, f \text{ then } f \in [\![\texttt{Nat}]\!] \Rightarrow S, \text{ and if } t \to^* \texttt{suc}\, u \text{ then } u \in S\}$ [5].

Similarly, the following encoding of the simply-typed λ-calculus is not PFP but can be proved accessible by taking

$$[\![\texttt{T}\ c]\!] = \text{if } c\!\downarrow = \texttt{arrow}\, a\, b \text{ then } \{t \in \mathrm{SN} \mid \text{if } t \to^* \texttt{lam} f \text{ then } f \in [\![\texttt{T}\ a]\!] \Rightarrow [\![\texttt{T}\ b]\!]\} \text{ else SN}$$

▶ **Example 15** (Simply-typed λ-calculus).

```
symbol Sort : TYPE                symbol arrow : Sort ⇒ Sort ⇒ Sort

symbol T : Sort ⇒ TYPE
  symbol lam : ∀ a b, (T a ⇒ T b) ⇒ T (arrow a b)
  symbol app : ∀ a b, T (arrow a b) ⇒ T a ⇒ T b
  rule app a b (lam _ _ f) x → f x
```

6 Size-change termination

In this section, we give a sufficient condition for $\tilde{>}$ to terminate. For first-order rewriting, many techniques have been developed for that purpose. To cite just a few, see for instance [17, 14]. Many of them can probably be extended to $\lambda\Pi/\mathcal{R}$, either because the structure of terms in which they are expressed can be abstracted away, or because they can be extended to deal also with variable applications, λ-abstractions and β-reductions.

As an example, following Wahlstedt [37], we are going to use Lee, Jones and Ben-Amram's size-change termination criterion (SCT) [26]. It consists in following arguments along function calls and checking that, in every potential loop, one of them decreases. First introduced for first-order functional languages, it has then been extended to many other settings: untyped λ-calculus [21], a subset of OCaml [32], Martin-Löf's type theory [37], System F [27].

We first recall Hyvernat and Raffalli's matrix-based presentation of SCT [20]:

▶ **Definition 16** (Size-change termination). *Let $\rhd$ be the smallest transitive relation such that $f t_1 \dots t_n \rhd t_i$ when $f \in \mathbb{F}$. The call graph $\mathcal{G}(\mathcal{R})$ associated to $\mathcal{R}$ is the directed labeled graph on the defined symbols of $\mathbb{F}$ such that there is an edge between f and g iff there is a dependency pair $f l_1 \dots l_p > g m_1 \dots m_q$. This edge is labeled with the matrix $(a_{i,j})_{i \le \mathrm{ar}(\Theta_f), j \le \mathrm{ar}(\Theta_g)}$ where:*

- *if $l_i \rhd m_j$, then $a_{i,j} = -1$;*
- *if $l_i = m_j$, then $a_{i,j} = 0$;*
- *otherwise $a_{i,j} = \infty$ (in particular if $i > p$ or $j > q$).*

$\mathcal{R}$ is size-change terminating (SCT) if, in the transitive closure of $\mathcal{G}(\mathcal{R})$ (using the min-plus semi-ring to multiply the matrices labeling the edges), all idempotent matrices labeling a loop have some -1 on the diagonal.

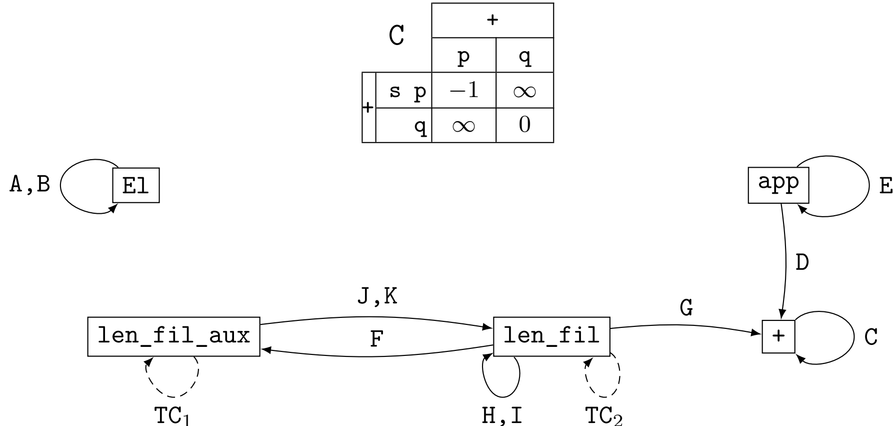


Figure 3 Matrix of dependency pair `C` and call graph of the dependency pairs of Example 7.

We add lines and columns of ∞'s in matrices associated to dependency pairs containing partially applied symbols (cases $i > p$ or $j > q$) because missing arguments cannot be compared with any other argument since they are arbitrary.

The matrix associated to the dependency pair `C: (s p) + q > p + q` and the call graph associated to the dependency pairs of Example 7 are depicted in Figure 3. The full list of matrices and the extensive call graph of Example 1 can be found in Appendix B.

▶ **Lemma 17.** *$\tilde{>}$ terminates if $\mathbb{F}$ is finite and $\mathcal{R}$ is SCT.*

Proof. Suppose that there is an infinite sequence $\chi = f_1\vec{t}_1\tilde{>}f_2\vec{t}_2\tilde{>}\ldots$ Then, there is an infinite path in the call graph going through nodes labeled by $f_1, f_2, \ldots$ Since $\mathbb{F}$ is finite, there is a symbol g occurring infinitely often in this path. So, there is an infinite sequence $g\vec{u}_1, g\vec{u}_2, \ldots$ extracted from χ. Hence, for every $i, j \in \mathbb{N}^*$, there is a matrix in the transitive closure of the graph which labels the loops of g corresponding to the relation between $\vec{u}_i$ and $\vec{u}_{i+j}$. By Ramsey's theorem, there is an infinite sequence (ϕ_i) and a matrix M such that M corresponds to all the transitions $g\vec{u}_{\phi_i}, g\vec{u}_{\phi_j}$ with $i \neq j$. M is idempotent, indeed $g\vec{u}_{\phi_i}, g\vec{u}_{\phi_{i+2}}$ is labeled by M^2 by definition of the transitive closure and by M due to Ramsey's theorem, so $M = M^2$. Since, by hypothesis, $\mathcal{R}$ satisfies SCT, there is j such that $M_{j,j}$ is -1. So, for all i, $u^{(j)}_{\phi_i}(\to^* \rhd)^+ u^{(j)}_{\phi_{i+1}}$. Since $\rhd\!\to \subseteq \to\!\rhd$ and $\to_{\text{arg}}$ is well-founded on $\mathbb{C}$, the existence of an infinite sequence contradicts the fact that $\rhd$ is well-founded. ◀

By combining all the previous results, we get:

▶ **Theorem 18.** *The relation $\to = \to_\beta \cup \to_{\mathcal{R}}$ terminates on terms typable in $\lambda\Pi/\mathcal{R}$ if $\to$ is locally confluent and preserves typing, $\mathbb{F}$ is finite and $\mathcal{R}$ is well-structured, plain-function passing and size-change terminating.*

The rewriting system of Example 1 verifies all these conditions (proof in the appendix).

7 Implementation and comparison with other criteria and tools

We implemented our criterion in a tool called SizeChangeTool [12]. As far as we know, there are no other termination checker for $\lambda\Pi/\mathcal{R}$.

If we restrict ourselves to simply-typed rewriting systems, then we can compare it with the termination checkers participating in the category "higher-order rewriting union beta" of the termination competition: Wanda uses dependency pairs, polynomial interpretations,

HORPO and many transformation techniques [24]; SOL uses the General Schema [6] and other techniques. As these tools implement various techniques and SizeChangeTool only one, it is difficult to compete with them. Still, there are examples that are solved by SizeChangeTool and not by one of the other tools, demonstrating that these tools would benefit from implementing our new technique. For instance, the problem `Hamana_Kikuchi_18/h17` is proved terminating by SizeChangeTool but not by Wanda because of the rule:

```
rule map f (map g l) → map (comp f g) l
```

And the problem `Kop13/kop12thesis_ex7.23` is proved terminating by SizeChangeTool but not by Sol because of the rules:[3]

```
rule f h x (s y) → g (c x (h y)) y    rule g x y → f (λ_,s 0) x y
```

One could also imagine to translate a termination problem in $\lambda\Pi/\mathcal{R}$ into a simply-typed termination problem. Indeed, the termination of $\lambda\Pi$ alone (without rewriting) can be reduced to the termination of the simply-typed λ-calculus [16]. This has been extended to $\lambda\Pi/\mathcal{R}$ when there are no type-level rewrite rules like the ones defining `El` in Example 1 [22]. However, this translation does not preserve termination as shown by the Example 15 which is not terminating if all the types $\mathbb{T}x$ are mapped to the same type constant.

In [30], Roux also uses dependency pairs for the termination of simply-typed higher-order rewriting systems, as well as a restricted form of dependent types where a type constant C is annotated by a pattern l representing the set of terms matching l. This extends to patterns the notion of indexed or sized types [18]. Then, for proving the absence of infinite chains, he uses simple projections [17], which can be seen as a particular case of SCT where strictly decreasing arguments are fixed (SCT can also handle permutations in arguments).

Finally, if we restrict ourselves to orthogonal systems, it is also possible to compare our technique to the ones implemented in the proof assistants Coq and Agda. Coq essentially implements a higher-order version of primitive recursion. Agda on the other hand uses SCT.

Because Example 1 uses matching on defined symbols, it is not orthogonal and can be written neither in Coq nor in Agda. Agda recently added the possibility of adding rewrite rules but this feature is highly experimental and comes with no guaranty. In particular, Agda termination checker does not handle rewriting rules.

Coq cannot handle inductive-recursive definitions [11] nor function definitions with permuted arguments in function calls while it is no problem for Agda and us.

8 Conclusion and future work

We proved a general modularity result extending Arts and Giesl's theorem that a rewriting relation terminates if there are no infinite sequences of dependency pairs [2] from first-order rewriting to dependently-typed higher-order rewriting. Then, following [37], we showed how to use Lee, Jones and Ben-Amram's size-change termination criterion to prove the absence of such infinite sequences [26].

This extends Wahlstedt's work [37] from weak to strong normalization, and from orthogonal to locally confluent rewriting systems. This extends the first author's work [5] from orthogonal to locally confluent systems, and from systems having a decreasing argument in each recursive call to systems with non-increasing arguments in recursive calls. Finally, this also extends previous works on static dependency pairs [25] from simply-typed λ-calculus to dependent types modulo rewriting.

[3] We renamed the function symbols for the sake of readability.

To get this result, we assumed local confluence. However, one often uses termination to check (local) confluence. Fortunately, there are confluence criteria not based on termination. The most famous one is (weak) orthogonality, that is, when the system is left-linear and has no critical pairs (or only trivial ones) [35], as it is the case in functional programming languages. A more general one is when critical pairs are "development-closed" [36].

This work can be extended in various directions.

First, our tool is currently limited to PFP rules, that is, to rules where higher-order variables are direct subterms of the left-hand side. To have higher-order variables in deeper subterms like in Example 14, we need to define a more complex interpretation of types, following [5].

Second, to handle recursive calls in such systems, we also need to use an ordering more complex than the subterm ordering when computing the matrices labeling the SCT call graph. The ordering needed for handling Example 14 is the "structural ordering" of Coq and Agda [9, 6]. Relations other than subterm have already been considered in SCT but in a first-order setting only [34].

But we may want to go further because the structural ordering is not enough to handle the following system which is not accepted by Agda:

▶ **Example 19** (Division). m/n computes $\lceil \frac{m}{n} \rceil$.

```
symbol minus: Nat⇒Nat⇒Nat            set infix 1 "-" ≔ minus
  rule 0 - n → 0      rule m - 0 → m      rule (s m) - (s n) → m - n
symbol div: Nat⇒Nat⇒Nat              set infix 1 "/" ≔ div
  rule 0 / (s n) → 0  rule (s m) / (s n) → s ((m - n) / (s n))
```

A solution to handle this system is to use arguments filterings (remove the second argument of `-`) or simple projections [17]. Another one is to extend the type system with size annotations as in Agda and compute the SCT matrices by comparing the size of terms instead of their structure [1, 7]. In our example, the size of `m - n` is smaller than or equal to the size of `m`. One can deduce this by using user annotations like in Agda, or by using heuristics [8].

Another interesting extension would be to handle function calls with locally size-increasing arguments like in the following example:

```
rule f x → g (s x)          rule g (s (s x)) → f x
```

where the number of `s`'s strictly decreases between two calls to `f` although the first rule makes the number of `s`'s increase. Hyvernat enriched SCT to handle such systems [19].

References

1 A. Abel. MiniAgda: integrating sized and dependent types. In *Proceedings of the Workshop on Partiality and Recursion in Interactive Theorem Provers*, Electronic Proceedings in Theoretical Computer Science 43, 2010. doi:10.4204/EPTCS.43.2.

2 T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000. doi:10.1016/S0304-3975(99)00207-8.

3 H. Barendregt. Lambda calculi with types. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of logic in computer science. Volume 2. Background: computational structures*, pages 117–309. Oxford University Press, 1992.

4 F. Blanqui. *Théorie des types et récriture*. PhD thesis, Université Paris-Sud, France, 2001. 144 pages. URL: http://tel.archives-ouvertes.fr/tel-00105522.

5 F. Blanqui. Definitions by rewriting in the calculus of constructions. *Mathematical Structures in Computer Science*, 15(1):37–92, 2005. doi:10.1017/S0960129504004426.

6 F. Blanqui. Termination of rewrite relations on λ-terms based on Girard's notion of reducibility. *Theoretical Computer Science*, 611:50–86, 2016. doi:10.1016/j.tcs.2015.07.045.

7 F. Blanqui. Size-based termination of higher-order rewriting. *Journal of Functional Programming*, 28(e11), 2018. 75 pages. doi:10.1017/S0956796818000072.

8 W. N. Chin and S. C. Khoo. Calculating sized types. *Journal of Higher-Order and Symbolic Computation*, 14(2-3):261–300, 2001. doi:10.1023/A:1012996816178.

9 T. Coquand. Pattern matching with dependent types. In *Proceedings of the International Workshop on Types for Proofs and Programs*, 1992. URL: http://www.lfcs.inf.ed.ac.uk/research/types-bra/proc/proc92.ps.gz.

10 D. Cousineau and G. Dowek. Embedding pure type systems in the lambda-Pi-calculus modulo. In *Proceedings of the 8th International Conference on Typed Lambda Calculi and Applications*, Lecture Notes in Computer Science 4583, 2007. URL: 10.1007/978-3-540-73228-0_9.

11 P. Dybjer. A general formulation of simultaneous inductive-recursive definitions in type theory. *Journal of Symbolic Logic*, 65(2):525–549, 2000. URL: http://www.jstor.org/stable/2586554.

12 G. Genestier. SizeChangeTool, 2018. URL: https://github.com/Deducteam/SizeChangeTool.

13 J. Giesl, R. Thiemann, and P. Schneider-Kamp. The dependency pair framework: combining techniques for automated termination proofs. In *Proceedings of the 11th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, Lecture Notes in Computer Science 3452, 2004. doi:10.1007/978-3-540-32275-7_21.

14 J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and improving dependency pairs. *Journal of Automated Reasoning*, 37(3):155–203, 2006. doi:10.1007/s10817-006-9057-7.

15 J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and types*. Cambridge University Press, 1988. URL: http://www.paultaylor.eu/stable/prot.pdf.

16 R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the ACM*, 40(1):143–184, 1993. doi:10.1145/138027.138060.

17 N. Hirokawa and A. Middeldorp. Tyrolean Termination Tool: techniques and features. *Information and Computation*, 205(4):474–511, 2007. doi:10.1016/j.ic.2006.08.010.

18 J. Hughes, L. Pareto, and A. Sabry. Proving the correctness of reactive systems using sized types. In *Proceedings of the 23th ACM Symposium on Principles of Programming Languages*, 1996. doi:10.1145/237721.240882.

19 P. Hyvernat. The size-change termination principle for constructor based languages. *Logical Methods in Computer Science*, 10(1):1–30, 2014. doi:10.2168/LMCS-10(1:11)2014.

20 P. Hyvernat and C. Raffalli. Improvements on the "size change termination principle" in a functional language. In *11th International Workshop on Termination*, 2010. URL: https://lama.univ-savoie.fr/~raffalli/pdfs/wst.pdf.

21 N. D. Jones and N. Bohr. Termination analysis of the untyped lambda-calculus. In *Proceedings of the 15th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 3091, 2004. doi:10.1007/978-3-540-25979-4_1.

22 J.-P. Jouannaud and J. Li. Termination of Dependently Typed Rewrite Rules. In *Proceedings of the 13th International Conference on Typed Lambda Calculi and Applications*, Leibniz International Proceedings in Informatics 38, 2015. doi:10.4230/LIPIcs.TLCA.2015.257.

23 J. W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory reduction systems: introduction and survey. *Theoretical Computer Science*, 121:279–308, 1993. doi:10.1016/0304-3975(93)90091-7.

24 C. Kop. *Higher order termination*. PhD thesis, VU University Amsterdam, 2012. URL: http://hdl.handle.net/1871/39346.

25 K. Kusakari and M. Sakai. Enhancing dependency pair method using strong computability in simply-typed term rewriting systems. *Applicable Algebra in Engineering Communication and Computing*, 18(5):407–431, 2007. doi:10.1007/s00200-007-0046-9.
26 C. S. Lee, N. D. Jones, and A. M. Ben-Amram. The size-change principle for program termination. In *Proceedings of the 28th ACM Symposium on Principles of Programming Languages*, 2001. doi:10.1145/360204.360210.
27 R. Lepigre and C. Raffalli. Practical subtyping for System F with sized (co-)induction, 2017. arXiv:1604.01990.
28 G. Markowsky. Chain-complete posets and directed sets with applications. *Algebra Universalis*, 6:53–68, 1976.
29 R. Mayr and T. Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192(2):3–29, 1998. doi:10.1016/S0304-3975(97)00143-6.
30 C. Roux. Refinement Types as Higher-Order Dependency Pairs. In *Proceedings of the 22nd International Conference on Rewriting Techniques and Applications*, Leibniz International Proceedings in Informatics 10, 2011. doi:10.4230/LIPIcs.RTA.2011.299.
31 R. Saillard. *Type checking in the Lambda-Pi-calculus modulo: theory and practice*. PhD thesis, Mines ParisTech, France, 2015. URL: https://pastel.archives-ouvertes.fr/tel-01299180.
32 D. Sereni and N. D. Jones. Termination analysis of higher-order functional programs. In *Proceedings of the 3rd Asian Symposium on Programming Languages and Systems*, Lecture Notes in Computer Science 3780, 2005. doi:10.1007/11575467_19.
33 TeReSe. *Term rewriting systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.
34 R. Thiemann and J. Giesl. The size-change principle and dependency pairs for termination of term rewriting. *Applicable Algebra in Engineering Communication and Computing*, 16(4):229–270, 2005. doi:10.1007/s00200-005-0179-7.
35 V. van Oostrom. *Confluence for abstract and higher-order rewriting*. PhD thesis, Vrije Universiteit Amsterdam, NL, 1994. URL: http://www.phil.uu.nl/~oostrom/publication/ps/phdthesis.ps.
36 V. van Oostrom. Developing developments. *Theoretical Computer Science*, 175(1):159–181, 1997. doi:10.1016/S0304-3975(96)00173-9.
37 D. Wahlstedt. *Dependent type theory with first-order parameterized data types and well-founded recursion*. PhD thesis, Chalmers University of Technology, Sweden, 2007. URL: http://www.cse.chalmers.se/alumni/davidw/wdt_phd_printed_version.pdf.

A Proofs of lemmas on the interpretation

A.1 Definition of the interpretation

▶ **Lemma 20.** *F is monotone wrt inclusion.*

Proof. We first prove that D is monotone. Let $I \subseteq J$ and $T \in D(I)$. We have to show that $T \in D(J)$. To this end, we have to prove (1) $T \in \mathrm{SN}$ and (2) if $T \to^* (x:A)B$ then $A \in \mathrm{dom}(J)$ and, for all $a \in J(A)$, $B[x \mapsto a] \in \mathrm{dom}(J)$:

1. Since $T \in D(I)$, we have $T \in \mathrm{SN}$.
2. Since $T \in D(I)$ and $T \to^* (x:A)B$, we have $A \in \mathrm{dom}(I)$ and, for all $a \in I(A)$, $B[x \mapsto a] \in \mathrm{dom}(I)$. Since $I \subseteq J$, we have $\mathrm{dom}(I) \subseteq \mathrm{dom}(J)$ and $J(A) = I(A)$ since I and J are functional relations. Therefore, $A \in \mathrm{dom}(J)$ and, for all $a \in I(A)$, $B[x \mapsto a] \in \mathrm{dom}(J)$.

We now prove that F is monotone. Let $I \subseteq J$ and $T \in D(I)$. We have to show that $F(I)(T) = F(J)(T)$. First, $T \in D(J)$ since D is monotone.

If $T\downarrow = (x:A)B$, then $F(I)(T) = \Pi a \in I(A).\, I(B[x \mapsto a])$ and $F(J)(T) = \Pi a \in J(A).\, J(B[x \mapsto a])$. Since $T \in D(I)$, we have $A \in \mathrm{dom}(I)$ and, for all $a \in I(A)$, $B[x \mapsto a] \in \mathrm{dom}(I)$. Since $\mathrm{dom}(I) \subseteq \mathrm{dom}(J)$, we have $J(A) = I(A)$ and, for all $a \in I(A)$, $J(B[x \mapsto a]) = I(B[x \mapsto a])$. Therefore, $F(I)(T) = F(J)(T)$.

Now, if $T\downarrow$ is not a product, then $F(I)(T) = F(J)(T) = \mathrm{SN}$. ◀

A.2 Computability predicates

▶ **Lemma 21.** *$\mathcal{D}$ is a computability predicate.*

Proof. Note that $\mathcal{D} = D(\mathcal{I})$.

1. $\mathcal{D} \subseteq \mathrm{SN}$ by definition of D.
2. Let $T \in \mathcal{D}$ and T' such that $T \to T'$. We have $T' \in \mathrm{SN}$ since $T \in \mathrm{SN}$. Assume now that $T' \to^* (x:A)B$. Then, $T \to^* (x:A)B$, $A \in \mathcal{D}$ and, for all $a \in \mathcal{I}(A)$, $B[x \mapsto a] \in \mathcal{D}$. Therefore, $T' \in \mathcal{D}$.
3. Let T be a neutral term such that $\to(T) \subseteq \mathcal{D}$. Since $\mathcal{D} \subseteq \mathrm{SN}$, $T \in \mathrm{SN}$. Assume now that $T \to^* (x:A)B$. Since T is neutral, there is $U \in \to(T)$ such that $U \to^* (x:A)B$. Therefore, $A \in \mathcal{D}$ and, for all $a \in \mathcal{I}(A)$, $B[x \mapsto a] \in \mathcal{D}$. ◀

▶ **Lemma 22.** *If $P \in \mathbb{P}$ and, for all $a \in P$, $Q(a) \in \mathbb{P}$, then $\Pi a \in P.\, Q(a) \in \mathbb{P}$.*

Proof. Let $R = \Pi a \in P.\, Q(a)$.

1. Let $t \in R$. We have to prove that $t \in \mathrm{SN}$. Let $x \in \mathbb{V}$. Since $P \in \mathbb{P}$, $x \in P$. So, $tx \in Q(x)$. Since $Q(x) \in \mathbb{P}$, $Q(x) \subseteq \mathrm{SN}$. Therefore, $tx \in \mathrm{SN}$, and $t \in \mathrm{SN}$.
2. Let $t \in R$ and t' such that $t \to t'$. We have to prove that $t' \in R$. Let $a \in P$. We have to prove that $t'a \in Q(a)$. By definition, $ta \in Q(a)$ and $ta \to t'a$. Since $Q(a) \in \mathbb{P}$, $t'a \in Q(a)$.
3. Let t be a neutral term such that $\to(t) \subseteq R$. We have to prove that $t \in R$. Hence, we take $a \in P$ and prove that $ta \in Q(a)$. Since $P \in \mathbb{P}$, we have $a \in \mathrm{SN}$ and $\to^*(a) \subseteq P$. We now prove that, for all $b \in \to^*(a)$, $tb \in Q(a)$, by induction on $\to$. Since t is neutral, tb is neutral too and it suffices to prove that $\to(tb) \subseteq Q(a)$. Since t is neutral, $\to(tb) = \to(t)b \cup t \to(b)$. By induction hypothesis, $t \to(b) \subseteq Q(a)$. By assumption, $\to(t) \subseteq R$. So, $\to(t)a \subseteq Q(a)$. Since $Q(a) \in \mathbb{P}$, $\to(t)b \subseteq Q(a)$ too. Therefore, $ta \in Q(a)$ and $t \in R$. ◀

▶ **Lemma 23.** *For all $T \in \mathcal{D}$, $\mathcal{I}(T)$ is a computability predicate.*

Proof. Since $\mathcal{F}_p(\mathbb{T}, \mathbb{P})$ is a chain-complete poset, it suffices to prove that $\mathcal{F}_p(\mathbb{T}, \mathbb{P})$ is closed by F. Assume that $I \in \mathcal{F}_p(\mathbb{T}, \mathbb{P})$. We have to prove that $F(I) \in \mathcal{F}_p(\mathbb{T}, \mathbb{P})$, that is, for all $T \in D(I)$, $F(I)(T) \in \mathbb{P}$. There are two cases:

- If $T\downarrow = (x:A)B$, then $F(I)(T) = \Pi a \in I(A).\, I(B[x \mapsto a])$. By assumption, $I(A) \in \mathbb{P}$ and, for $a \in I(A)$, $I(B[x \mapsto a]) \in \mathbb{P}$. Hence, by Lemma 22, $F(I)(T) \in \mathbb{P}$.
- Otherwise, $F(I)(T) = \mathrm{SN} \in \mathbb{P}$. ◀

▶ **Lemma 4a.** *For all terms T and substitutions σ, $[\![T]\!]_\sigma \in \mathbb{P}$.*

Proof. By induction on T. If $T = s$, then $[\![T]\!]_\sigma = \mathcal{D} \in \mathbb{P}$ by Lemma 21. If $T = (x:A)K \in \mathbb{K}$, then $[\![T]\!]_\sigma = \Pi a \in [\![A]\!]_\sigma.\, [\![K]\!]_{[x \mapsto a, \sigma]}$. By induction hypothesis, $[\![A]\!]_\sigma \in \mathbb{P}$ and, for all $a \in [\![A]\!]_\sigma$, $[\![K]\!]_{[x \mapsto a, \sigma]} \in \mathbb{P}$. Hence, by Lemma 22, $[\![T]\!]_\sigma \in \mathbb{P}$. If $T \notin \mathbb{K} \cup \{\mathrm{KIND}\}$ and $T\sigma \in \mathcal{D}$, then $[\![T]\!]_\sigma = \mathcal{I}(T\sigma) \in \mathbb{P}$ by Lemma 23. Otherwise, $[\![T]\!]_\sigma = \mathrm{SN} \in \mathbb{P}$. ◀

A.3 Invariance by reduction

We now prove that the interpretation is invariant by reduction.

▶ **Lemma 24.** *If $T \in \mathcal{D}$ and $T \to T'$, then $\mathcal{I}(T) = \mathcal{I}(T')$.*

Proof. First note that $T' \in \mathcal{D}$ since $\mathcal{D} \in \mathbb{P}$. Hence, $\mathcal{I}(T')$ is well defined. Now, we have $T \in \mathrm{SN}$ since $\mathcal{D} \subseteq \mathrm{SN}$. So, $T' \in \mathrm{SN}$ and, by local confluence and Newman's lemma, $T\!\downarrow = T'\!\downarrow$. If $T\!\downarrow = (x : A)B$ then $\mathcal{I}(T) = \Pi a \in \mathcal{I}(A). \mathcal{I}(B[x \mapsto a]) = \mathcal{I}(T')$. Otherwise, $\mathcal{I}(T) = \mathrm{SN} = \mathcal{I}(T')$. ◀

▶ **Lemma 4b.** *If T is typable, $T\sigma \in \mathcal{D}$ and $T \to T'$, then $[\![T]\!]_\sigma = [\![T']\!]_\sigma$.*

Proof. By assumption, there are Γ and U such that $\Gamma \vdash T : U$. Since $\to$ preserves typing, we also have $\Gamma \vdash T' : U$. So, $T \neq \mathrm{KIND}$, and $T' \neq \mathrm{KIND}$. Moreover, $T \in \mathbb{K}$ iff $T' \in \mathbb{K}$ since $\Gamma \vdash T : \mathrm{KIND}$ iff $T \in \mathbb{K}$ and T is typable. In addition, we have $T'\sigma \in \mathcal{D}$ since $T\sigma \in \mathcal{D}$ and $\mathcal{D} \in \mathbb{P}$.

We now prove the result, with $T \to^= T'$ instead of $T \to T'$, by induction on T. If $T \notin \mathbb{K}$, then $T' \notin \mathbb{K}$ and, since $T\sigma, T'\sigma \in \mathcal{D}$, $[\![T]\!]_\sigma = \mathcal{I}(T\sigma) = \mathcal{I}(T'\sigma) = [\![T']\!]_\sigma$ by Lemma 24. If $T = \mathrm{TYPE}$, then $[\![T]\!]_\sigma = \mathcal{D} = [\![T']\!]_\sigma$. Otherwise, $T = (x : A)K$ and $T' = (x : A')K'$ with $A \to^= A'$ and $K \to^= K'$. By inversion, we have $\Gamma \vdash A : \mathrm{TYPE}$, $\Gamma \vdash A' : \mathrm{TYPE}$, $\Gamma, x : A \vdash K : \mathrm{KIND}$ and $\Gamma, x : A' \vdash K' : \mathrm{KIND}$. So, by induction hypothesis, $[\![A]\!]_\sigma = [\![A']\!]_\sigma$ and, for all $a \in [\![A]\!]_\sigma$, $[\![K]\!]_{\sigma'} = [\![K']\!]_{\sigma'}$, where $\sigma' = [x \mapsto a, \sigma]$. Therefore, $[\![T]\!]_\sigma = [\![T']\!]_\sigma$. ◀

▶ **Lemma 4c.** *If T is typable, $T\sigma \in \mathcal{D}$ and $\sigma \to \sigma'$, then $[\![T]\!]_\sigma = [\![T]\!]_{\sigma'}$.*

Proof. By induction on T.

- If $T \in \mathbb{S}$, then $[\![T]\!]_\sigma = \mathcal{D} = [\![T]\!]_{\sigma'}$.
- If $T = (x : A)K$ and $K \in \mathbb{K}$, then $[\![T]\!]_\sigma = \Pi a \in [\![A]\!]_\sigma. [\![K]\!]_{[x \mapsto a, \sigma]}$ and $[\![T]\!]_{\sigma'} = \Pi a \in [\![A]\!]_{\sigma'}. [\![K]\!]_{[x \mapsto a, \sigma']}$. By induction hypothesis, $[\![A]\!]_\sigma = [\![A]\!]_{\sigma'}$ and, for all $a \in [\![A]\!]_\sigma$, $[\![K]\!]_{[x \mapsto a, \sigma]} = [\![K]\!]_{[x \mapsto a, \sigma']}$. Therefore, $[\![T]\!]_\sigma = [\![T]\!]_{\sigma'}$.
- If $T\sigma \in \mathcal{D}$, then $[\![T]\!]_\sigma = \mathcal{I}(T\sigma)$ and $[\![T]\!]_{\sigma'} = \mathcal{I}(T\sigma')$. Since $T\sigma \to^* T\sigma'$, by Lemma 4b, $\mathcal{I}(T\sigma) = \mathcal{I}(T\sigma')$.
- Otherwise, $[\![T]\!]_\sigma = \mathrm{SN} = [\![T]\!]_{\sigma'}$. ◀

A.4 Adequacy of the interpretation

▶ **Lemma 4d.** *If $(x : A)B$ is typable, $((x : A)B)\sigma \in \mathcal{D}$ and $x \notin \mathrm{dom}(\sigma) \cup \mathrm{FV}(\sigma)$, then $[\![(x : A)B]\!]_\sigma = \Pi a \in [\![A]\!]_\sigma. [\![B]\!]_{[x \mapsto a, \sigma]}$.*

Proof. If B is a kind, this is immediate. Otherwise, since $((x : A)B)\sigma \in \mathcal{D}$, $[\![(x : A)B]\!]_\sigma = \mathcal{I}(((x : A)B)\sigma)$. Since $x \notin \mathrm{dom}(\sigma) \cup \mathrm{FV}(\sigma)$, we have $((x : A)B)\sigma = (x : A\sigma)B\sigma$. Since $(x : A\sigma)B\sigma \in \mathcal{D}$ and $\mathcal{D} \subseteq \mathrm{SN}$, we have $[\![(x : A)B]\!]_\sigma = \Pi a \in \mathcal{I}(A\sigma\!\downarrow). \mathcal{I}((B\sigma\!\downarrow)[x \mapsto a])$.

Since $(x : A)B$ is typable, A is of type TYPE and $A \notin \mathbb{K} \cup \{\mathrm{KIND}\}$. Hence, $[\![A]\!]_\sigma = \mathcal{I}(A\sigma)$ and, by Lemma 24, $\mathcal{I}(A\sigma) = \mathcal{I}(A\sigma\!\downarrow)$.

Since $(x : A)B$ is typable and not a kind, B is of type TYPE and $B \notin \mathbb{K} \cup \{\mathrm{KIND}\}$. Hence, $[\![B]\!]_{[x \mapsto a, \sigma]} = \mathcal{I}(B[x \mapsto a, \sigma])$. Since $x \notin \mathrm{dom}(\sigma) \cup \mathrm{FV}(\sigma)$, $B[x \mapsto a, \sigma] = (B\sigma)[x \mapsto a]$. Hence, $[\![B]\!]_{[x \mapsto a, \sigma]} = \mathcal{I}((B\sigma)[x \mapsto a])$ and, by Lemma 24, $\mathcal{I}((B\sigma)[x \mapsto a]) = \mathcal{I}((B\sigma\!\downarrow)[x \mapsto a])$.

Therefore, $[\![(x : A)B]\!]_\sigma = \Pi a \in [\![A]\!]_\sigma. [\![B]\!]_{[x \mapsto a, \sigma]}$. ◀

Note that, by iterating this lemma, we get that $v \in [\![\forall\vec{x}:\vec{T},U]\!]$ iff, for all $\vec{t}$ such that $[\vec{x}\mapsto\vec{t}] \models \vec{x}:\vec{T}$, $v\vec{t} \in [\![U]\!]_{[\vec{x}\mapsto\vec{t}]}$.

▶ **Lemma 4e.** *If $\Delta \vdash U : s$, $\Gamma \vdash \gamma : \Delta$ and $U\gamma\sigma \in \mathcal{D}$, then $[\![U\gamma]\!]_\sigma = [\![U]\!]_{\gamma\sigma}$.*

Proof. We proceed by induction on U. Since $\Delta \vdash U : s$ and $\Gamma \vdash \gamma : \Delta$, we have $\Gamma \vdash U\gamma : s$.

- If $s = \text{TYPE}$, then $U, U\gamma \notin \mathbb{K} \cup \{\text{KIND}\}$ and $[\![U\gamma]\!]_\sigma = \mathcal{I}(U\gamma\sigma) = [\![U]\!]_{\gamma\sigma}$ since $U\gamma\sigma \in \mathcal{D}$.
- Otherwise, $s = \text{KIND}$ and $U \in \mathbb{K}$.
  - If $U = \text{TYPE}$, then $[\![U\gamma]\!]_\sigma = \mathcal{D} = [\![U]\!]_{\gamma\sigma}$.
  - Otherwise, $U = (x : A)K$ and, by Lemma 4d, $[\![U\gamma]\!]_\sigma = \Pi a \in [\![A\gamma]\!]_\sigma.\, [\![K\gamma]\!]_{[x\mapsto a,\sigma]}$ and $[\![U]\!]_{\gamma\sigma} = \Pi a \in [\![A]\!]_{\gamma\sigma}.\, [\![K]\!]_{[x\mapsto a,\gamma\sigma]}$. By induction hypothesis, $[\![A\gamma]\!]_\sigma = [\![A]\!]_{\gamma\sigma}$ and, for all $a \in [\![A\gamma]\!]_\sigma$, $[\![K\gamma]\!]_{[x\mapsto a,\sigma]} = [\![K]\!]_{\gamma[x\mapsto a,\sigma]}$. Wlog we can assume $x \notin \mathrm{dom}(\gamma) \cup \mathrm{FV}(\gamma)$. So, $[\![K]\!]_{\gamma[x\mapsto a,\sigma]} = [\![K]\!]_{[x\mapsto a,\gamma\sigma]}$. ◀

▶ **Lemma 4f.** *Let P be a computability predicate and Q a P-indexed family of computability predicates such that $Q(a') \subseteq Q(a)$ whenever $a \to a'$. Then, $\lambda x : A.b \in \Pi a \in P.\, Q(a)$ whenever $A \in \mathrm{SN}$ and, for all $a \in P$, $b[x \mapsto a] \in Q(a)$.*

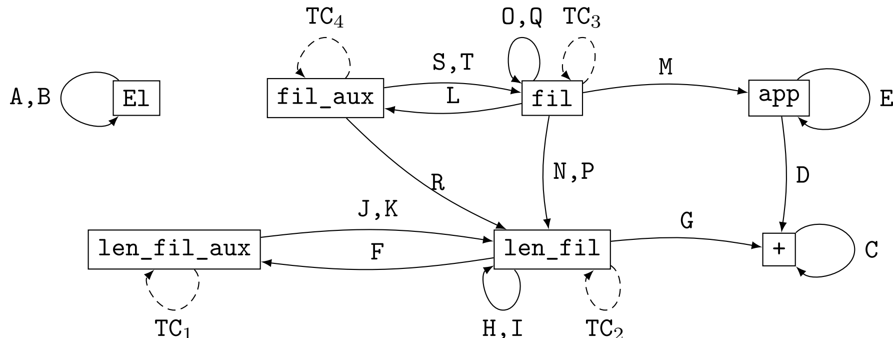
Proof. Let $a_0 \in P$. Since $P \in \mathbb{P}$, we have $a_0 \in \mathrm{SN}$ and $x \in P$. Since $Q(x) \in \mathbb{P}$ and $b = b[x \mapsto x] \in Q(x)$, we have $b \in \mathrm{SN}$. Let $a \in \to^* (a_0)$. We can prove that $(\lambda x : A, b)a \in Q(a_0)$ by induction on (A, b, a) ordered by $(\to, \to, \to)_\forall$:.,Since $Q(a_0) \in \mathbb{P}$ and $(\lambda x : A, b)a$ is neutral, it suffices to prove that $\to((\lambda x : A, b)a) \subseteq Q(a_0)$. If the reduction takes place in A, b or a, we can conclude by induction hypothesis. Otherwise, $(\lambda x : A, b)a \to b[x \mapsto a] \in Q(a)$ by assumption. Since $a_0 \to^* a$ and $Q(a') \subseteq Q(a)$ whenever $a \to a'$, we have $b[x \mapsto a] \in Q(a_0)$. ◀

B Termination proof of Example 1

Here is the comprehensive list of dependency pairs in the example:

```
A:               El (arrow a b) > El a
B:               El (arrow a b) > El b
C:                    (s p) + q > p + q
D:   app a _ (cons _ x p l) q m > p + q
E:   app a _ (cons _ x p l) q m > app a p l q m
F:len_fil a f _ (cons _ x p l)  > len_fil_aux (f x) a f p l
G:len_fil a f _ (app _ p l q m) >
                        (len_fil a f p l) + (len_fil a f q m)
H:len_fil a f _ (app _ p l q m) > len_fil a f p l
I:len_fil a f _ (app _ p l q m) > len_fil a f q m
J:    len_fil_aux true  a f p l > len_fil a f p l
K:    len_fil_aux false a f p l > len_fil a f p l
L:    fil a f _ (cons _ x p l)  > fil_aux (f x) a f x p l
M:    fil a f _ (app _ p l q m) >
                        app a (len_fil a f p l) (fil a f p l)
                              (len_fil a f q m) (fil a f q m)
N:    fil a f _ (app _ p l q m) > len_fil a f p l
O:    fil a f _ (app _ p l q m) > fil a f p l
P:    fil a f _ (app _ p l q m) > len_fil a f q m
Q:    fil a f _ (app _ p l q m) > fil a f q m
R:      fil_aux true  a f x p l > len_fil a f p l
S:      fil_aux true  a f x p l > fil a f p l
T:      fil_aux false a f x p l > fil a f p l
```

The whole callgraph is depicted below. The letter associated to each matrix corresponds to the dependency pair presented above and in example 7, except for TC 's which comes from the computation of the transitive closure and labels dotted edges.



The argument `a` is omitted everywhere on the matrices presented below:

$$\texttt{A,B}=\begin{pmatrix}-1\end{pmatrix},\ \texttt{C}=\begin{pmatrix}-1 & \infty \\ \infty & 0\end{pmatrix},\ \texttt{D}=\begin{pmatrix}\infty & \infty \\ -1 & \infty \\ \infty & 0 \\ \infty & \infty\end{pmatrix},\ \texttt{E}=\begin{pmatrix}\infty & \infty & \infty & \infty \\ -1 & -1 & \infty & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 0\end{pmatrix},\ \texttt{F}=\begin{pmatrix}\infty & 0 & \infty & \infty \\ \infty & \infty & \infty & \infty \\ \infty & \infty & -1 & -1\end{pmatrix},\ \texttt{J}=\texttt{K}=\begin{pmatrix}\infty & \infty & \infty \\ 0 & \infty & \infty \\ \infty & 0 & \infty \\ \infty & \infty & 0\end{pmatrix},$$

$$\texttt{G}=\begin{pmatrix}\infty & \infty \\ \infty & \infty \\ \infty & \infty\end{pmatrix},\ \texttt{H}=\texttt{I}=\texttt{N}=\texttt{O}=\texttt{P}=\texttt{Q}=\begin{pmatrix}0 & \infty & \infty \\ \infty & \infty & \infty \\ \infty & -1 & -1\end{pmatrix},\ \texttt{L}=\begin{pmatrix}\infty & 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & -1 & -1 & -1\end{pmatrix},\ \texttt{M}=\begin{pmatrix}\infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty\end{pmatrix},$$

$$\texttt{R}=\texttt{S}=\texttt{T}=\begin{pmatrix}\infty & \infty & \infty \\ 0 & \infty & \infty \\ \infty & \infty & \infty \\ \infty & 0 & \infty \\ \infty & \infty & 0\end{pmatrix}.$$

Which leads to the matrices labeling a loop in the transitive closure:

$$\texttt{TC}_1=\texttt{J}\times\texttt{F}=\begin{pmatrix}\infty & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & \infty \\ \infty & \infty & -1 & -1\end{pmatrix},\ \texttt{TC}_4=\texttt{S}\times\texttt{L}=\begin{pmatrix}\infty & \infty & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & -1 & -1 & -1\end{pmatrix},$$

$$\texttt{TC}_3=\texttt{L}\times\texttt{S}=\texttt{TC}_2=\texttt{F}\times\texttt{J}=\begin{pmatrix}0 & \infty & \infty \\ \infty & \infty & \infty \\ \infty & -1 & -1\end{pmatrix}=\texttt{O}=\texttt{H}.$$

It would be useless to compute matrices labeling edges which are not in a strongly connected component of the call-graph (like S×R), but it is necessary to compute all the products which could label a loop, especially to verify that all loop-labeling matrices are idempotent, which is indeed the case here.

We now check that this system is well-structured. For each rule $f\vec{l} \to r$, we take the environment $\Delta_{f\vec{l}\to r}$ made of all the variables of r with the following types: `a:Set, b:Set, p:ℕ, q:ℕ, x:El a, l:𝕃 a p, m:𝕃 a q, f:El a⇒ 𝔹`.

The precedence infered for this example is the smallest containing:

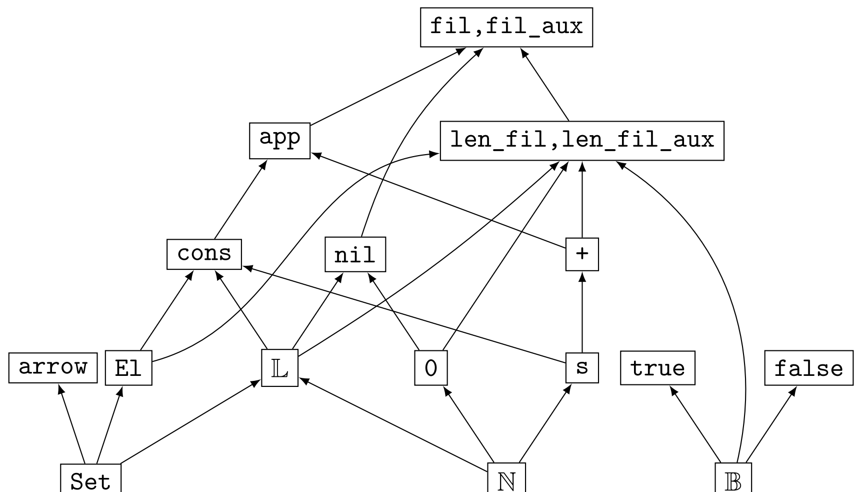
- comparisons linked to the typing of symbols:

$$\begin{array}{rcl@{\qquad}rcl}
\texttt{Set} & \preceq & \texttt{arrow} & \texttt{Set},\mathbb{L},\texttt{0} & \preceq & \texttt{nil} \\
\texttt{Set} & \preceq & \texttt{El} & \texttt{Set},\texttt{El},\mathbb{N},\mathbb{L},\texttt{s} & \preceq & \texttt{cons} \\
\mathbb{B} & \preceq & \texttt{true} & \texttt{Set},\mathbb{N},\mathbb{L},\texttt{+} & \preceq & \texttt{app} \\
\mathbb{B} & \preceq & \texttt{false} & \texttt{Set},\texttt{El},\mathbb{B},\mathbb{N},\mathbb{L} & \preceq & \texttt{len_fil} \\
\mathbb{N} & \preceq & \texttt{0} & \mathbb{B},\texttt{Set},\texttt{El},\mathbb{N},\mathbb{L} & \preceq & \texttt{len_fil_aux} \\
\mathbb{N} & \preceq & \texttt{s} & \texttt{Set},\texttt{El},\mathbb{B},\mathbb{N},\mathbb{L},\texttt{len_fil} & \preceq & \texttt{fil} \\
\mathbb{N} & \preceq & \texttt{+} & \mathbb{B},\texttt{Set},\texttt{El},\mathbb{N},\mathbb{L},\texttt{len_fil_aux} & \preceq & \texttt{fil_aux} \\
\texttt{Set},\mathbb{N} & \preceq & \mathbb{L} & & &
\end{array}$$

- and comparisons related to calls:

$$\begin{array}{rcl@{\qquad}rcl}
\texttt{s} & \preceq & \texttt{+} & \texttt{s},\texttt{len_fil} & \preceq & \texttt{len_fil_aux} \\
\texttt{cons},\texttt{+} & \preceq & \texttt{app} & \texttt{nil},\texttt{fil_aux},\texttt{app},\texttt{len_fil} & \preceq & \texttt{fil} \\
\texttt{0},\texttt{len_fil_aux},\texttt{+} & \preceq & \texttt{len_fil} & \texttt{fil},\texttt{cons},\texttt{len_fil} & \preceq & \texttt{fil_aux}
\end{array}$$

This precedence can be sum up in the following diagram, where symbols in the same box are equivalent:



A Generic Framework for Higher-Order Generalizations

David M. Cerna
FMV and RISC, Johannes Kepler University Linz, Austria
david.cerna@risc.jku.at

Temur Kutsia
RISC, Johannes Kepler University Linz, Austria
kutsia@risc.jku.at

Abstract

We consider a generic framework for anti-unification of simply typed lambda terms. It helps to compute generalizations which contain maximally common top part of the input expressions, without nesting generalization variables. The rules of the corresponding anti-unification algorithm are formulated, and their soundness and termination are proved. The algorithm depends on a parameter which decides how to choose terms under generalization variables. Changing the particular values of the parameter, we obtained four new unitary variants of higher-order anti-unification and also showed how the already known pattern generalization fits into the schema.

2012 ACM Subject Classification Theory of computation → Rewrite systems; Theory of computation → Higher order logic; Theory of computation → Type theory

Keywords and phrases anti-unification, typed lambda calculus, least general generalization

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.10

Funding Supported by the Austrian Science Fund (FWF) under the project P 28789-N32.

Acknowledgements We thank Tomer Libal for useful discussions on the early version of the paper.

1 Introduction

A term r is generalization of a term t, if t can be obtained from r by a variable substitution. The problem of finding common generalizations of two or more terms has been investigated quite intensively. The main idea is to compute least general generalizations (lggs) which maximally keep the similarities between the input terms and uniformly abstract over differences in them by new variables. For instance, if the input terms are $t = f(a, a)$ and $s = f(b, b)$, we are interested in their lgg $f(x, x)$. It gives more precise information about the nature of t and s than their other generalizations such as, e.g., $f(x, y)$ or just x. Namely, it shows that t and s not only have the same head f, but also each of them has its both arguments equal.

The technique of computing generalizations is called anti-unification. It was introduced in 1970s [17, 18] and saw a renewed interest in recent years (see, e.g., [3, 2, 11, 6, 1]), mostly motivated by various applications (see, e.g., [5, 13, 19, 20]).

Concerning anti-unification for higher-order terms, lggs are not unique and special fragments or variants of the problem have to be considered to guarantee uniqueness of lggs. Such special cases include generalizations with higher-order patterns [8, 7, 15, 16], object terms [9], restricted terms [21], etc. For instance, a pattern lgg of $\lambda x.f(g(x))$ and $\lambda x.h(g(x))$ is $\lambda x.Y(x)$, ignoring the fact that those terms have a common subterm $g(x)$. It happens because the pattern restriction requires free variables to apply to sequences of distinct bound variables. That's why we get a generalization in which the free variable Y applies to the bound variable x, and not to the more complex common subterm $g(x)$ of the given terms.

© David M. Cerna and Temur Kutsia;
licensed under Creative Commons License CC-BY
4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019).
Editor: Herman Geuvers; Article No. 10; pp. 10:1–10:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Pattern generalizations have been successfully used, e.g., in term indexing [16] and in software code analysis for quick bug fixing [19]. Another advantage is that they can be computed efficiently, in linear time [8]. However, some problems require more expressive variants than patterns, as, for instance, the application of higher-order anti-unification in automatic detection of recursion schemes in functional programming [5]. Combination of increased expressive power and good computational properties was the motivation behind the introduction of functions-as-constructors terms (fc-terms) in higher-order unification [12]. We address a similar problem for anti-unification in this paper.

One difficulty comes from the fact that not all lggs are good characterizations of generalized terms. For instance, for $\lambda x.f(g(x))$ and $\lambda x.f(h(x))$ both $\lambda x.X(f(g(x)), f(h(x)))$ and $\lambda x.f(Y(g(x), h(x)))$ are lggs (where X and Y are fresh generalization variables), but the latter one is better, since it shows that the input terms have the common head f. This observation leads to the notion of top-maximal generalization, which keeps the maximally large common top part of the input terms. This is a very natural property, which, by default, was guaranteed for pattern lggs, but not necessarily for lggs in richer variants.

Another problem is related with nested generalization variables, which may affect least generality. In practice, such a nesting causes nondeterminism, an undesirable property. Once a difference between terms is detected, it should be abstracted by a generalization variable and no attempt to further generalize should be made under it. This leads to a variant of generalization, which we call shallow with respect to generalization variables, since these variables are not nested. We concentrate on computing top-maximal shallow lggs.

In this setting the interesting question is, what terms are permitted under generalization variables? For the pattern case they are distinct bound variables, but we want more expressive variants. Instead of coming up with different particular cases and designing special anti-unification algorithms for them, we formulate a generic algorithm which always computes top-maximal shallow generalizations, and prove its soundness and termination (Sect. 4). The particular cases can be obtained by instantiating a parameter in one of the rules of the algorithm, which is responsible for choosing terms under generalization variables. Changing the specific values of the parameter, we obtained several new unitary (i.e., with single lgg) variants of higher-order anti-unification: projection-based (Sect. 5.1), generalizations with common subterms (Sect. 5.2.1), relaxed fc (Sect. 5.2.2), and fc (Sect. 5.2.3). We also show how pattern generalization fits into the schema (Sect. 5.2.4). Completeness results for these variants are given. Additional variants of higher-order anti-unification can be developed using our schema by specifying how terms under generalization variables are chosen. For reader's convenience, some illustrative examples are put in the appendix.

2 Preliminaries

We consider simply-typed signature, where *types* are constructed from a set of *basic types* (denoted by δ) by the grammar $\tau ::= \delta \mid \tau \to \tau$, where $\to$ is associative to the right. *Variables* (denoted by $X, Y, Z, x, y, z, \ldots$) and *constants* (denoted by $a, b, c, f, g, h, \ldots$) have an assigned type. The set of variables is denoted by V and the set of constants by C. λ-*terms* (denoted by $t, s, r, \ldots$) are built using the grammar $t ::= x \mid c \mid \lambda x.t \mid t_1\ t_2$ where x is a variable and c is a constant, and are typed as usual. Terms of the form $(\ldots(h\ t_1)\ldots t_m)$, where h is a constant or a variable, will be written as $h(t_1, \ldots, t_m)$, and terms of the form $\lambda x_1.\cdots.\lambda x_n.t$ as $\lambda x_1, \ldots, x_n.t$. We use $\vec{x}$ as a short-hand for $x_1, \ldots, x_n$.

Other standard notions of the simply typed λ-calculus, like bound and free occurrences of variables, subterms, α-conversion, β-reduction, η-long β-normal form, etc. are defined as usual (see, e.g., [4]). $t{\downarrow_\eta}$ denotes the η-normal form of t. We denote the fact that t is a (strict) subterm of s using the infix binary symbol $\sqsubseteq$ ($\sqsubset$). Bound variables will be denoted

by lowercase letters and free variables by capital letters. The symbols $\mathsf{fv}(t)$ and $\mathsf{bv}(t)$ are used to denote the sets of free and bound variables, respectively, of a term t. This notation extends to a set of terms as well. A term t is *closed* if $\mathsf{fv}(t) = \emptyset$.

By default, terms are assumed to be written in η-long β-normal form. Therefore, all terms have the form $\lambda x_1, \dots, x_n.h(t_1, \dots, t_m)$, where $n, m \geq 0$, h is either a constant or a variable, $t_1, \dots, t_m$ also have this form, and the term $h(t_1, \dots, t_m)$ has a basic type. For a term $t = \lambda x_1, \dots, x_n.h(t_1, \dots, t_m)$ with $n, m \geq 0$, its *head* is defined as $head(t) = h$.

When we write an equality between two λ-terms, we mean α, β and η equivalence.

Positions in λ-terms are defined with respect to their tree representation in the usual way, as string of integers. For instance, in the term $f(\lambda x.\lambda y.g(\lambda z.h(z, y), x), \lambda u.g(u))$, the symbol f stands in the position ϵ (the empty sequence), the occurrence of $\lambda x.$ stands in the position 1, the bound occurrence of y in 1.1.1.1.1.2, the bound occurrence of u in 2.1.1, etc. Hence, abstractions in this context are treated as symbols. We denote the symbol occurring in position p in a term t by $\mathsf{symb}(t, p)$ and the subterm of t at position p by $t|_p$. We write $p_1 \leq p_2$ if the position p_1 is a prefix of p_2. The strict part of this ordering is denoted by $<$. The set of all positions of a term t is denoted by $Pos(t)$.

Substitutions and their composition ($\circ$) are defined as usual. Namely, $(\sigma \circ \vartheta)X = \vartheta(\sigma(X))$. 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 $\sigma \preceq \vartheta$, if there exists a substitution φ such that $\sigma \circ \varphi = \vartheta$. The strict part of this relation is denoted by $\prec$. The relation $\preceq$ is a partial order and generates the equivalence relation which we denote by $\simeq$. We overload $\preceq$ by defining $s \preceq t$ if there exists a substitution σ such that $s\sigma = t$.

3 Special forms of terms, generalization problems

In this section we first introduce certain special forms of terms and then discuss the generalization problem where the generalization terms may be of a special restricted form.

▶ **Definition 1** (Restricted terms). *Let B be a set of variables and s be a term such that $B \cap \mathsf{bv}(s) = \emptyset$. Assume that distinct bound variables have distinct names in s. We say that a term t is* B-restricted *in s if t is a subterm of s such that* (i) *t is η-equivalent to some $t' \in B \cup \mathsf{bv}(s)$, or* (ii) *$t = (f\, t_1 \cdots t_n)$, where $n > 0$, $f \in C \cup B \cup \mathsf{bv}(s)$ and each t_i, $1 \leq i \leq n$, is a B-restricted term in s.*

▶ **Definition 2** (Relaxed functions-as-constructors triples). *Let F and B be two disjoint sets of variables and s be a term such that $F \cap \mathsf{bv}(s) = B \cap \mathsf{bv}(s) = \emptyset$. It is also assumed that distinct bound variables have distinct names in s. We say that the triple (F, B, s) is a* relaxed functions-as-constructors triple *or, shortly,* rfc-triple*, if the following conditions are satisfied:*

Argument restriction: *For all occurrences of $(X\, t_1 \cdots t_n)$ in s, where $X \in F$, t_i is a B-restricted term in s for each $0 < i \leq n$.*

Local restriction: *For all occurrences of $(X\, t_1 \cdots t_n)$ in s, where $X \in F$, for each $0 < i, j \leq n$, if $i \neq j$, then $t_i{\downarrow_\eta} \not\sqsubseteq t_j{\downarrow_\eta}$.*

Functions-as-constructors triples are rfc-triples obeying a global restriction:

▶ **Definition 3** (Functions-as-constructors triples). *Let F, B, and s be as in Definition 2 and (F, B, s) be an rfc-triple. We say that it is a* functions-as-constructors triple *or, shortly,* fc-triple*, if the following extra condition is satisfied:*

Global restriction: *For each two different occurrences of terms $(X\, t_1 \cdots t_n)$ and $(Y\, s_1 \cdots s_m)$ in s with $X, Y \in F$, for each $0 < i \leq n$, $0 < j \leq m$, we have $t_i{\downarrow_\eta} \not\sqsubset s_j{\downarrow_\eta}$.*

▶ **Definition 4** (Pattern triples). *Let F, B, and s be as in Definition 2 and $(\mathsf{F}, \mathsf{B}, s)$ be an rfc-triple. A triple is* pattern *if the following stronger form of the argument restriction holds:*

Argument restriction for patterns: *For all occurrences of $(X\, t_1 \cdots t_n)$ in s, where $X \in \mathsf{F}$, we have $t_i{\downarrow_\eta} \in \mathsf{B} \cup \mathsf{bv}(s)$ for each $0 < i \leq n$.*

Note that for patterns the local restriction reduces to checking whether $t_i{\downarrow_\eta}$'s are distinct bound variables, and the global restriction is automatically fulfilled. Hence, pattern triples are also a special case of fc-triples.

▶ **Definition 5** (Rfc-terms, fc-terms, patterns, shallow terms). *Let F be a set of variables and t be a term such that $\mathsf{F} \cap \mathsf{bv}(t) = \emptyset$. Then t is an* F-rfc-term *(resp.,* F-fc-term*,* F-pattern*), if $(\mathsf{F}, \emptyset, t)$ is an rfc-triple (resp., fc-triple, pattern-triple). We say that t is an* F-shallow term *if for every subterm $(X\, t_1 \cdots t_n)$ of t with $X \in \mathsf{F}$, we have $\mathsf{F} \cap (\cup_{i=1}^n \mathsf{fv}(t_i)) = \emptyset$.*

A term t is shallow*, if it is an $\mathsf{fv}(t)$-shallow term. Rfc-terms, fc-terms and patterns are defined analogously.*

Note that pattern coincides with the well-known higher-order patterns [14] fragment. Every pattern is an fc-term. Every fc-term is an rfc-term. Every rfc-term is a shallow term.

▶ **Example 6.** Consider the following terms:

$$t_1 = \lambda x, y.\, f(X(x, \lambda z_1.y(z_1)),\ Y(\lambda z_2.y(z_2), x),\ \lambda u.Z(x, u))$$
$$t_2 = \lambda x, y.\, f(X(g(x), p(\lambda z_1.y(z_1), \lambda z_2.y(z_2))), h(Y(g(x), g(h(x))))),$$
$$t_3 = \lambda x.\, f(X(g(x), h(x)), h(Y(g(x), g(h(x))))),$$
$$t_4 = \lambda x.\, f(X(x, g(a)), x), \qquad t_5 = \lambda x.\, f(X(x, g(x)), x), \qquad t_6 = \lambda x, y.\, f(X(Y(x), y)).$$

t_1 is a pattern; t_2 is an fc-term but not a pattern; t_3 is an rfc-term but not an fc-term; t_4 is a shallow term but not an rfc-term, the argument restriction is violated; t_5 is a shallow term but not an rfc-term, the local restriction is violated; t_6 is not a shallow term.

A term t is called a *generalization* or an *anti-instance* of two terms t_1 and t_2 if $t \preceq t_1$ and $t \preceq t_2$. It is the *least general generalization* (lgg), also known as a *most specific anti-instance,* of t_1 and t_2, if there is no generalization s of t_1 and t_2 which satisfies $t \prec s$.

An *anti-unification triple* (shortly AUT) has the form $X(\vec{x}) : t \triangleq s$ where $\lambda\vec{x}.X(\vec{x})$, $\lambda\vec{x}.t$, and $\lambda\vec{x}.s$ are terms of the same type, t and s are in η-long β-normal form, and X does not occur in t and s. An *anti-unifier* of an AUT $X(\vec{x}) : t \triangleq s$ is a substitution σ such that $dom(\sigma) = \{X\}$ and $\lambda\vec{x}.X(\vec{x})\sigma$ is a term which generalizes both $\lambda\vec{x}.t$ and $\lambda\vec{x}.s$.

An anti-unifier σ of $X(\vec{x}) : t \triangleq s$ is *least general* (or *most specific*) if there is no anti-unifier ϑ of the same problem that satisfies $\sigma \prec \vartheta$. Obviously, if σ is a least general anti-unifier of an AUT $X(\vec{x}) : t \triangleq s$, then $\lambda\vec{x}.X(\vec{x})\sigma$ is a lgg of $\lambda\vec{x}.t$ and $\lambda\vec{x}.s$.

If r is a generalization of t and s, the *set of generalization variables (genvars)* of t and s in r is the set $\mathsf{genvar}(r, t, s) := \mathsf{fv}(r) \setminus (\mathsf{fv}(s) \cup \mathsf{fv}(t))$.

Our main interest is in generalizations, which retain the common parts of the given terms as much as possible, at least until the first differences in each branch during top-down traversal of the given terms. This intuition is formalized in the following definition:

▶ **Definition 7** (Top-maximal generalization). *Let s and t be terms of the same type in η-long β-normal form such that the bound variables are renamed uniformly: the ith bound variable in depth-first pre-order traversal in s and in t have the same name x_i. A common generalization r of s and t is their top-maximal common generalization, if the following conditions hold:*

- *If* $\mathsf{symb}(s,\epsilon) = \mathsf{symb}(t,\epsilon)$, *then* $\mathsf{symb}(r,\epsilon) = \mathsf{symb}(s,\epsilon)$.
- *If* $p \in Pos(s) \cap Pos(t)$ *such that* $\mathsf{symb}(s,q) = \mathsf{symb}(t,q)$ *for all positions* $q < p$ *and* $\mathsf{symb}(s,p) = \mathsf{symb}(t,p)$, *then* $p \in Pos(r)$ *and* $\mathsf{symb}(r,p) = \mathsf{symb}(s,p)$.
- *If* $p \in Pos(s) \cap Pos(t)$ *such that* $\mathsf{symb}(s,q) = \mathsf{symb}(t,q)$ *for all positions* $q < p$ *and* $\mathsf{symb}(s,p) \neq \mathsf{symb}(t,p)$, *then* $p \in Pos(r)$ *and* $\mathsf{symb}(r,p)$ *is a genvar.*

▶ **Example 8.** Let $s = \lambda x.f(g(x))$ and $t = \lambda x.f(h(x))$. Then $r_1 = \lambda x.X(f(g(x)), f(h(x)))$ is their shallow but not top-maximal generalization, while $r_2 = \lambda x.f(Y(g(x), h(x)))$ is both top-maximal and shallow. Also, $r_3 = \lambda x.f(Z(x))$ is a top-maximal shallow generalization.

3.1 Variants of higher-order anti-unification

In the literature, a *variant* of a unification or anti-unification problem is obtained by imposing restrictions on the form of solutions to the problem (in contrast to *fragments*, where the form of input is restricted). Here we define variants of higher-order anti-unification problem, which we will be solving in the coming sections.

The main variant we consider is what we call the top-maximal genvar-shallow variant:

Given: Two terms t and s of the same type in η-long β-normal form.
Find: A top-maximal generalization r of t and s such that r is a $\mathsf{genvar}(r,t,s)$-shallow term.

The problem statement implies that we are looking for r which is least general among all top-maximal genvar-shallow generalizations of t and s. There can still exist a term which is less general than r, is a top-maximal generalization of both s and t, but is not a genvar-shallow term. Also, there can exist a genvar-shallow generalization of s and t which is less general than r, but it is not a top-maximal generalization of s and t.

By imposing various conditions on r, we get other special problems such as, cs (common subterms), rfc, fc, and pattern variants of higher-order anti-unification.

First, we define the cs-variant. We need an auxiliary definition of extension of a set of terms by variables (bound in the context):

▶ **Definition 9.** *Let* B *be a set of variables and* S *be a set of terms such that* $\mathsf{bv}(S) \cap \mathsf{B} = \emptyset$. *By* B-extension *of* S *we understand the set* $S \cup (\mathsf{B} \setminus \mathsf{fv}(S))$.

Now, the definition of generalization with common subterms can be formulated as follows:

▶ **Definition 10** (CS-generalization)**.** *A generalization* r *of two terms* s *and* t *is called their* common-subterms generalization, *shortly* cs-generalization, *if it is a* $\mathsf{genvar}(r,s,t)$*-shallow top-maximal generalization of* t *and* s *satisfying the following condition:*

Common subterms condition: *Let* p *be a position in* r, $r|_p = X(r_1,\dots,r_n)$ *for a genvar* X *and some terms* $r_1,\dots,r_n$, *and* B *be the set of all variables bound by* λ *at positions above* p, *i.e.,* $\mathsf{B} := \{x \mid \mathsf{symb}(r,q) = \lambda x$ *for some position* $q < p\}$. *Then* $\{r_1,\dots,r_n\}$ *is the* B'*-extension of some set of common subterms of* $s|_p$ *and* $t|_p$, *where* $\mathsf{B}' \subseteq \mathsf{B}$.

A cs-generalization r *of* s *and* t *is their least general cs-generalization (cs-lgg) if no cs-generalization* r' *of* s *and* t *satisfies* $r \prec r'$.

In this definition, top-maximality guarantees that all positions $q < p$ in r are also positions in t and s and $\mathsf{symb}(r,q) = \mathsf{symb}(t,q) = \mathsf{symb}(s,q)$ (modulo α-renaming of t and s). Therefore it may well happen that variables from B appear in $t|_p$ or in $s|_p$. Since r is a generalization, $r_1,\dots,r_n$ must contain all variables from B that appear in $t|_p$ or in $s|_p$, for otherwise one can not get $t|_p$ and $s|_p$ from $r|_p$ by a substitution for X. Hence, actually, we have $\mathsf{B} \cap (\mathsf{fv}(t|_p) \cup \mathsf{fv}(s|_p)) \subseteq \mathsf{B}' \subseteq \mathsf{B}$ in Definition 10.

The cs-variant is the problem of computing cs-generalizations. The rfc, fc, and pattern variants are defined similarly, based on the following definition of the corresponding generalizations:

▶ **Definition 11** (RFC-, FC-, pattern-generalizations). *A generalization r of s and t is their rfc-generalization if r is an rfc- term. It is a least general rfc-generalization (rfc-lgg) of s and t if no rfc-generalization r' of s and t satisfies $r \prec r'$. The rfc-variant of higher-order anti-unification s the problem of computing rfc-generalizations. Fc- and pattern generalizations, lggs, and variants are defined in the same way.*

▶ **Example 12.** We bring examples of various lggs and show how they related to each other. Let $t = \lambda x.f(h(g(g(x))), h(g(x)), a)$ and $s = \lambda x.f(g(g(x)), g(x), h(a))$. Then

- $r_0 = \lambda x.f(X(h(g(g(x))), g(g(x))), X(h(g(x)), g(x)), X(a, h(a)))$ is a shallow top-maximal lgg of t and s.
- $r_1 = \lambda x.f(X(g(g(x))), X(g(x)), Z(a))$ is a cs-lgg of t and s. We have $r_1 \prec r_0$.
- $r_2 = \lambda x.f(X(g(g(x))), X(g(x)), Z)$ is a top-maximal rfc-lgg of t and s. We have $r_2 \prec r_1$.
- $r_3 = \lambda x.f(X(g(x)), Y(g(x)), Z)$ is a top-maximal fc-lgg of t and s and $r_3 \prec r_2$.
- $r_4 = \lambda x.f(X(x), Y(x), Z)$ is a top-maximal pattern-lgg of t and s. Also here $r_4 \prec r_3$.

More precise relationships between cs, rfc, fc, and pattern variants will be investigated in Section 5.2 below.

Top-maximality is an important requirement for an lgg to exist. If we do not require it, we might have $\preceq$-incomparable generalizations. For instance, in Example 8, r_1 and r_2 are not comparable by $\preceq$ and r_1 and r_3 are not either. On the other hand, two top-maximal shallow generalizations r_2 and r_3 are: $r_3 \prec r_2$.

For patterns, top-maximality means also least generality. This is not the case for shallow terms, as Example 8 shows. In fact, from that example we can see that top-maximality does not imply least generality for fc- and rfc-generalizations either, because r_1 and r_2 are both fc- and rfc-generalizations of s and t.

4 Generic anti-unification transformation rules

Transformation rules for anti-unification work on triples $A; S; r$, which we call *states*. Here A is a set of AUTs of the form $\{X_1(\overrightarrow{x_1}) : t_1 \triangleq s_1, \dots, X_n(\overrightarrow{x_n}) : t_n \triangleq s_n\}$ that are pending to anti-unify, S is a set of already solved AUTs (the *store*), and r is a generalization (computed so far). The goal is, given two terms t and s, compute a generalization r which is a $\mathsf{genvar}(r, t, s)$-shallow term. We aim at computing lggs.

The transformation rules given below are generic. At first, they help us to obtain a top-maximal genvar-shallow generalization. From it, we can obtain more special generalizations (e.g., rfc, fc, patterns) by deciding which kind of arguments are allowed under genvars.

▶ Remark 13. We assume that in the set $A \cup S$ each occurrence of λ binds a distinct name variable and that each generalization variable occurs in $A \cup S$ only once.

The set of transformations $\mathfrak{G}$ is defined by the following rules:

Dec: **Decomposition**

$$\{X(\vec{x}) : h(t_1, \dots, t_m) \triangleq h(s_1, \dots, s_m)\} \uplus A;\ S;\ r \Longrightarrow$$
$$\{Y_1(\vec{x}) : t_1 \triangleq s_1, \dots, Y_m(\vec{x}) : t_m \triangleq s_m\} \cup A;\ S;\ r\{X \mapsto \lambda\vec{x}.h(Y_1(\vec{x}), \dots, Y_m(\vec{x}))\},$$

where $Y_1, \dots, Y_n$ are fresh variables of the appropriate types.

Abs: **Abstraction**

$\{X(\vec{x}) : \lambda y.t \triangleq \lambda z.s\} \uplus A;\ S;\ r \Longrightarrow$
$\quad \{X'(\vec{x}, y) : t \triangleq s\{z \mapsto y\}\} \cup A;\ S;\ r\{X \mapsto \lambda\vec{x}, y.X'(\vec{x}, y)\}.$

where X' is a fresh variable of the appropriate type.

Sol: **Solve**

$\{X(\vec{x}) : t \triangleq s\} \uplus A;\ S;\ r \Longrightarrow$
$\quad A;\ \{Y(y_1, \dots, y_n) : (C_t\, y_1 \cdots y_n) \triangleq (C_s\, y_1 \cdots y_n)\} \cup S;\ r\{X \mapsto \lambda\vec{x}.Y(q_1, \dots, q_n)\},$

where t and s are of a basic type, $head(t) \neq head(s)$, $q_1, \dots, q_n$ are distinct subterms of t or s, C_t and C_s are terms such that $(C_t\, q_1 \cdots q_n) = t$ and $(C_s\, q_1 \cdots q_n) = s$, C_t and C_s do not contain any $x \in \vec{x}$, and Y, $y_1, \dots, y_n$ are distinct fresh variables of the appropriate type.

Mer: **Merge**

$\emptyset;\ \{X(\vec{x}) : t_1 \triangleq s_1, Y(\vec{y}) : t_2 \triangleq s_2\} \uplus S;\ r \Longrightarrow \emptyset;\ \{X(\vec{x}) : t_1 \triangleq s_1\} \cup S;\ r\{Y \mapsto \lambda\vec{y}.X(\vec{x}\pi)\},$

where $\pi : \{\vec{x}\} \to \{\vec{y}\}$ is a bijection, extended as a substitution, with $t_1\pi = t_2$ and $s_1\pi = s_2$.

To compute generalizations for t and s, we start with the *initial state* $\{X : t \triangleq s\}; \emptyset; X$, where X is a fresh variable, and apply the transformations as long as possible. These *final states* have the form $\emptyset; S; r$, Then, the *result computed* by $\mathfrak{G}$ is r.

We use the letters C_t and C_s in the Solve rule because these terms resemble multi-contexts. Each of them have a form $\lambda z_1, \dots, z_n.C'_t$ and $\lambda z_1, \dots, z_n.C'_s$, where the bound variables $z_1, \dots, z_n$ play the role of holes. In the store we keep the η-long β-normal form of $(C_t\, y_1 \cdots y_n)$ and $(C_s\, y_1 \cdots y_n)$. When applied to $q_1, \dots, q_n$, C_t and C_s give, respectively, t and s. However, it should be emphasized that it is not the choice of C_t and C_s that might cause branching applications of Sol, but the choice of the subterms $q_1, \dots, q_n$. Moreover, choosing different special forms of $q_1, \dots, q_n$, we obtain different special versions of the anti-unification algorithm.

One can easily show that rules map a state to a state: For each expression $X(\vec{x}) : t \triangleq s \in A \cup S$, the terms $X(\vec{x})$, t and s have the same type, s and t are in η-long β-normal form, and X does not occur in t and s. Moreover, all genvars are distinct.

The property that each occurrence of λ in $A \cup S$ binds a unique variable is also maintained. It guarantees that in the Abs rule, the variable y is fresh for s. After the application of the rule, y will appear nowhere else in $A \cup S$ except $X'(\vec{x}, y)$ and, maybe, t and s.

▶ **Theorem 14.** *Let t and s be terms. Any sequence of transformations in $\mathfrak{G}$ starting from the initial state $\{X : t \triangleq s\}; \emptyset; X$ terminates and each computed result r is a* genvar(r, t, s)*-shallow top-maximal generalization of t and s.*

Proof. Let the size of an AUT $Z(\vec{z}) : p \triangleq q$ be the number of symbols occurring in p or q, and the size of a set of AUTs be the multiset of sizes of AUTs it contains. Then the first three rules in $\mathfrak{G}$ strictly reduce the size of A. Mer applies when A is empty and strictly reduces the size of S. Hence, the algorithm terminates. The computed result is an genvar(r, t, s)-shallow term, since no rule puts one generalization variable on top of another.

Proving that a computed result is a generalization is more involved. First, we prove that if $A_1; S_1; r \Longrightarrow A_2; S_2; r\vartheta$ is one step, then for any $X(\vec{x}) : t \triangleq s \in A_1 \cup S_1$, we have $X(\vec{x})\vartheta \preceq t$ and $X(\vec{x})\vartheta \preceq s$. Note that if $X(\vec{x}) : t \triangleq s$ was not transformed at this step, then this property trivially holds for it. Therefore, we assume that $X(\vec{x}) : t \triangleq s$ is selected and prove the property for each rule. We only illustrate it for Sol here, for the other rules the proof proceeds as in [8].

Sol: We have $\vartheta = \{X \mapsto \lambda\vec{x}.Y(q_1,\ldots,q_n)\}$, where $q_1,\ldots,q_n$ are distinct subterms in t or s. Let $\psi_1 = \{Y \mapsto \lambda y_1,\ldots,y_n.(C_t\, y_1 \cdots y_n)\}$ and $\psi_2 = \{Y \mapsto \lambda y_1,\ldots,y_n.(C_s\, y_1 \cdots y_n)\}$. Since $(C_t\, q_1 \cdots q_n) = t$, $(C_s\, q_1 \cdots q_n) = s$, and C_t and C_s do not contain any variable $x \in \vec{x}$, we get $X(\vec{x})\vartheta\psi_1 = X(\vec{x})\{X \mapsto \lambda\vec{x}.t,\ldots\} = t$, $X(\vec{x})\vartheta\psi_2 = X(\vec{x})\{X \mapsto \lambda\vec{x}.s,\ldots\} = s$, and, hence, $X(\vec{x})\vartheta \preceq t$ and $X(\vec{x})\vartheta \preceq s$.

We proceed by induction on the length l of the transformation sequence. We will prove a more general statement: If $A_0; S_0; r\vartheta_0 \Longrightarrow^* \emptyset; S_n; r\vartheta_0\vartheta_1 \cdots \vartheta_n$ is a transformation sequence in $\mathfrak{G}$, then for any $X(\vec{x}) : t \triangleq s \in A_0 \cup S_0$ we have $X(\vec{x})\vartheta_1 \cdots \vartheta_n \preceq t$ and $X(\vec{x})\vartheta_1 \cdots \vartheta_n \preceq s$.

When $l = 1$, it is exactly the one-step case we just proved. Assume that the statement is true for any transformation sequence of the length n and prove it for a transformation sequence $A_0; S_0; \vartheta_0 \Longrightarrow A_1; S_1; \vartheta_0\vartheta_1 \Longrightarrow^* \emptyset; S_n; \vartheta_0\vartheta_1 \cdots \vartheta_n$ of the length $n+1$.

Below the composition $\vartheta_i\vartheta_{i+1}\cdots\vartheta_k$ is abbreviated as ϑ_i^k with $k \geq i$. Let $X(\vec{x}) : t \triangleq s$ be an AUT selected for transformation at the current step. (Again, the property trivially holds for the AUTs which are not selected). We have to consider each rule, but, like above, only Sol is illustrated. For the other rules the proof is similar to the one in [8].

Sol: We have $X(\vec{x})\vartheta_1^1 = Y(q_1,\ldots,q_n)$ where Y is in the store. By the induction hypothesis, $Y(q_1,\ldots,q_n)\vartheta_2^n \preceq t$ and $Y(q_1,\ldots,q_n)\vartheta_2^n \preceq s$. Therefore, $X(\vec{x})\vartheta_1^n \preceq t$ and $X(\vec{x})\vartheta_1^n \preceq s$.

Finally, note that the obtained generalization is top-maximal, because the algorithm proceeds inserting common top-parts of the input terms in the generalization term as much as possible, and introduces a generalization variable only when a difference is encountered. ◀

▶ **Corollary 15.** *The result computed by $\mathfrak{G}$ for closed terms s and t is a shallow top-maximal generalization of s and t.*

As one can notice, the store keeps track of the differences between the original terms and suggests how to obtain them from the generalization. If the computed generalization for t and s is $\lambda\vec{x}.Y(q_1,\ldots,q_n)$ and the store contains the AUT $Y(y_1,\ldots,y_n) : (C_t\, y_1 \cdots y_n) \triangleq (C_s\, y_1 \cdots y_n)$, the substitution $\{Y \mapsto \lambda y_1,\ldots,y_n.(C_t\, y_1 \cdots y_n)\}$ gives t and $\{Y \mapsto \lambda y_1,\ldots,y_n.(C_s\, y_1 \cdots y_n)\}$ gives s.

▶ **Theorem 16** (Uniqueness modulo $\simeq$). *Assume that the set $\{q_1,\ldots,q_n\}$, C_t, and C_s in the Solve rule are uniquely determined (modulo renaming of bound variables). Assume that for a given t' and s', $\mathfrak{G}$ can compute their generalizations r_1 and r_2 with different sequence of rule applications. Then $r_1 \simeq r_2$.*

Proof. In [8] it was proved that different order of the Mer rule application gives equivalent solutions, provided that the other rules are applied in a unique way to the selected AUT. The same for Abs and Dec rules. Sol can be applied also only in one way, since $\{q_1,\ldots,q_n\}$, C_t, and C_s are uniquely determined. Therefore, the theorem follows from Theorem 4 in [8]. ◀

4.1 The Solve rule

The Solve rule is generic and leaves room for special versions of the algorithm depending on how the subterms $q_1,\ldots,q_n$ are chosen. The choice of C_t and C_s is also important since they might affect applicability of the Merge rule. To illustrate the latter, consider the generalization derivation for the terms $\lambda x.f(g(x,a), g(a,x))$ and $\lambda x.f(h(x,a), h(a,x))$:

$$\{\lambda x.f(g(x,a), g(a,x)) \triangleq \lambda x.f(h(x,a), h(a,x))\};\ \emptyset;\ X \Longrightarrow_{\mathsf{Abs,\ Dec}}$$

$$\{X_1(x) : g(x,a) \triangleq h(x,a),\ X_2(x) : g(a,x) \triangleq h(a,x)\};\ \emptyset;\ \lambda x.f(X_1(x), X_2(x)) \Longrightarrow_{\mathsf{Sol}}$$

$$\{X_2(x) : g(a,x) \triangleq h(a,x)\};\ \{Y(y_1,y_2) : g(y_1,y_2) \triangleq h(y_1,y_2)\};$$
$$\lambda x.f(Y(x,a), X_2(x)) \Longrightarrow_{\mathsf{Sol}}$$
$$\emptyset;\ \{Y(y_1,y_2) : g(y_1,y_2) \triangleq h(y_1,y_2),\ Z(z_1,z_2) : g(a,z_1) \triangleq h(z_2,z_1)\};$$
$$\lambda x.f(Y(x,a), Z(x,a)).$$

Here we chose C_t, C_s terms differently in two applications of Sol: First time, we had $q_1 = x$, $q_2 = a$ and we replaced in $t = g(x,a)$ and $s = h(x,a)$ all occurrences of the q's by fresh variables. Second time, in $t = g(a,x)$ and $s = h(a,x)$, we again took $q_1 = x$, $q_2 = a$, but the occurrence of q_2 in t is not replaced by a new variable. It resulted into the terminal store. However, the obtained generalization is not an lgg. An lgg would be $\lambda x.f(Y(x,a), Y(a,x))$.

If in the second application of Sol we again replaced all occurrences of the q's by fresh variables, we would make the step, leading to the mentioned lgg:

$$\emptyset;\ \{Y(y_1,y_2) : g(y_1,y_2) \triangleq h(y_1,y_2), Z(z_1,z_2) : g(z_2,z_1) \triangleq h(z_2,z_1)\};$$
$$\lambda x.f(Y(x,a), Z(x,a)) \Longrightarrow_{\mathsf{Mer}}$$
$$\emptyset;\ \{Y(y_1,y_2) : g(y_1,y_2) \triangleq h(y_1,y_2)\};\ \lambda x.f(Y(x,a), Y(a,x)).$$

Now we will formulate general rules for choosing the subterms $q_1,\dots,q_n$ and terms C_t and C_s in Sol. The rules will depend on a generic selection function. The function chooses subterms that satisfy a condition allowing them to appear under genvars. Our goal is to show that if $q_1,\dots,q_n$, C_t, and C_s in Sol are chosen according to the rules, then the computed generalization is least general among all similar generalizations.

Note that the condition of Sol implies that $q_1,\dots,q_n$ contain all variables from $\vec{x}$ that appear in t or in s, and contain none from $\vec{x}$ that appear neither in t nor in s.

We call (p_1,p_2) an *extended position pair* if p_1 and p_2 are either positions (positive integer sequences), or p_1 is a position and $p_2 = \bullet$ (a special symbol), or $p_1 = \bullet$ and p_2 is a position. The symbol $\bullet$ is not comparable with any position with respect to prefix ordering $\leq$. The latter is extended to pairs componentwise: $(p_1,p_2) \leq (l_1,l_2)$ iff $p_i \leq l_i$, $i \in \{1,2\}$. Then for its strict part, $(p_1,p_2) < (l_1,l_2)$ iff either $p_1 < l_1$ and $p_2 \leq l_2$, or $p_1 \leq l_1$ and $p_2 < l_2$.

Given two terms t_1 and t_2, a *triple of subterm occurrence* in t_1 or t_2 is a triple (p_1,p_2,s) where (p_1,p_2) is an extended position pair such that

- if $p_i \in Pos(t_i)$, $i \in \{1,2\}$, then $s = t_1|_{p_1} = t_2|_{p_2}$,
- if $p_1 \in Pos(t_1)$ and $p_2 = \bullet$, then $s = t_1|_{p_1}$ and s does not occur in t_2,
- if $p_1 = \bullet$ and $p_2 \in Pos(t_2)$, then $s = t_2|_{p_2}$ and s does not occur in t_1.

Now we define a selection function which will be used to define the ways we could select the terms $q_1,\dots,q_n$ in the Sol rule. It will depend on a special condition, a parameter, whose specific values will give specific variants of higher order generalizations in the next sections.

▶ **Definition 17.** *Given a set of variables $\{\vec{x}\}$ and terms t_1 and t_2, the* Select *function with the parametric condition* cond*, $\mathsf{Select}_{\mathsf{cond}}(\{\vec{x}\}, t_1, t_2)$, is the set of all subterm occurrence triples $Q = \{(p_1^1, p_1^2, s_1), \dots, (p_k^1, p_k^2, s_k)\}$ in t_1 or in t_2 such that*

1. $\mathsf{cond}(\{\vec{x}\}, t_1, t_2, Q)$ *holds.*
2. *If a variable from $\{\vec{x}\}$ appears in position p in t_1 (resp. in t_2), then there exist $p' \leq p$ and l such that $(p', l, t_1|_{p'}) \in Q$ (resp. $(l, p', t_2|_{p'}) \in Q$).*
3. *For all $(p_1,p_2,s) \in Q$, there is no $(p_1',p_2',s') \in Q$ such that $(p_1',p_2') < (p_1,p_2)$ holds.*

Now we define general rules for choosing $q_1,\dots,q_n$, C_t, and C_s in Sol. Let $\{X(\vec{x}) : t \triangleq s\} \uplus A$ be the set of AUTs on which Sol operates. Let $\mathsf{Select}_{\mathsf{cond}}(\{\vec{x}\}, t, s) = Q$. The rule of choosing $q_1,\dots,q_n$ in Sol is the following:

QR: $\{q_1, \dots, q_n\} = \{q \mid (p_1, p_2, q) \in Q$ for some p_1 and $p_2\}$.

For the rule for C_t and C_s, we will need a special notation. By $t[p \mapsto x]$ we denote a term obtained from t by replacing its subterm at position p by the variable x. If the position p does not exist in t, or if $p = \bullet$, then $t[p \mapsto x] = t$.

CR: Let $Q = \{(p_{i1}, l_{i1}, q_i), \dots, (p_{ik}, l_{ik}, q_i) \mid 1 \le i \le n\}$. Then:

$$C_t = \lambda y_1, \dots, y_n.\, t[p_{11} \mapsto y_1] \cdots [p_{1k_1} \mapsto y_1] \cdots [p_{n1} \mapsto y_n] \cdots [p_{nk_n} \mapsto y_n]$$
$$C_s = \lambda y_1, \dots, y_n.\, s[l_{11} \mapsto y_1] \cdots [l_{1k_1} \mapsto y_1] \cdots [l_{n1} \mapsto y_n] \cdots [l_{nk_n} \mapsto y_n].$$

CR says that no eligible occurrence of each q_i is kept in C_t and C_s. In those positions, if they still exist in C_t and C_s, we have the variable y_i.

The next step is to define special cases of the generic algorithm by specifying cond.

5 Special cases

The special cases of the generic algorithm are obtained by deciding what kind of subterms from the input terms we would like to preserve in the generalization under genvars.

We distinguish between two classes of (top-maximal, genvar-shallow) generalizations:

(a) Those which do not care about common subterms under different-head terms to be generalized but, rather, take both different-head terms entirely in the generalization, and

(b) Those which try to find similarities under different-head terms to be generalized and select their certain common subterms to the generalization.

We call the first class *projection-based variant*, since the generalizations there give original terms by projection substitutions. The second class corresponds to the *common-subterm variant*, introduced earlier. There are several subcategories in this class, as we will see.

5.1 Projection-based variant

This is the simplest case. If t and s appear in the Sol rule, we should keep both of them in the generalization. Therefore, we specify Select and, consequently, **QR** and **CR** as follows:

Specifying $\text{Select}_{\text{cond}}(\{\vec{x}\}, t, s)$: cond is always true.
The instance of QR: $q_1 = t$, $q_2 = s$.
The instance of CR: $C_t = \lambda z_1, z_2.\, z_1$, $C_s = \lambda z_1, z_2.\, z_2$.

After applying Sol with $\text{Select}_{\text{cond}}$ specified above, the new AUT in the store will have the form $Y(y_1, y_2) : y_1 \triangleq y_2$. By the exhaustive application of the Mer rule we get that if the computed result contains genvars, then it contains only one such variable (maybe with multiple occurrences). Therefore, we can ignore Mer and use the same variable. The store is not needed at all, since merging is superfluous and the anti-unifiers are fixed to projections. The instance of Sol rule is denoted by Sol-PrB, and the obtained algorithm by $\mathfrak{G}_{\text{prb}}$.

Generalizations that retain both terms whose heads are different are called imitation-free generalizations in [10] (where only second-order generalizations are considered), motivated from [9]. The name originates from the fact that one does not need imitation anti-unifiers. We prefer the name projection-based, since it directly indicates how the anti-unifiers look.

▶ **Theorem 18.** *$\mathfrak{G}_{\text{prb}}$ computes a projection-based genvar(r, t, s)-shallow top-maximal generalization r of the input terms t and s in linear time.*

Proof. Top-maximality and shallowness of r follow from Theorem 14, the projection-based property from the instances on **QR** and **CR**, and the linear time complexity from the fact that each symbol in the input is processed only once, when it is put into the generalization. ◀

▶ **Theorem 19** (Completeness of $\mathfrak{G}_{\text{prb}}$). *If r_0 is a projection-based $\mathsf{genvar}(r_0, t, s)$-shallow top-maximal generalization of t and s, then $\mathfrak{G}_{\text{prb}}$ computes r, starting from t and s, such that $r_0 \preceq r$.*

Proof sketch. Top maximal projection-based genvar-shallow generalizations of t and s can differ from each other only by the number of duplicates among genvars. $\mathfrak{G}_{\text{prb}}$ maximizes their sharing. Hence, r is less general than any projection-based genvar-shallow generalization. ◀

▶ **Corollary 20.** *Projection-based variant of higher-order anti-unification is unitary.*

Proof. Follows from Theorem 19 and Theorem 16, since the specification of instances of **QR** and **CR** makes the q's and C's in the Solve rule uniquely determined. ◀

Interestingly, projection-based generalizations are least general among all top-maximal generalizations that do not nest genvars:

▶ **Theorem 21.** *Let r_1 and r_2 be respectively $\mathsf{genvar}(r_1, t, s)$- and $\mathsf{genvar}(r_2, t, s)$-shallow top-maximal generalizations of t and s. Assume that r_1 is projection-based. Then $r_2 \preceq r_1$.*

Proof. By the definition of projection-based generalization, the symbols occurring above the positions of genvars are common for t and s. Top-maximality requires that common symbols are retained in the generalization. Let r_1 and r_2 contain a genvar in position p. Since they are top-maximal, all symbols above p are common in s and t. Since r_1 is $\mathsf{genvar}(r_1, t, s)$-shallow and projection-based, $r_1|_p$ should have a form $Y(t|_p, s|_p)$, where Y is a genvar.

Also, r_2 is $\mathsf{genvar}(r_1, t, s)$-shallow. Therefore, $r_2|_p$ has a form $X(q_1, \dots, q_n)$, where each q_i is a subterm of $t|_p$ or $s|_p$. Since r_2 is a generalization of t and s, there exist substitutions σ_1 and σ_2 such that $X(q_1, \dots, q_n)\sigma_1 = t|_p$ and $X(q_1, \dots, q_n)\sigma_2 = s|_p$. Then $r_1|_p$ can be obtained from $r_2|_p$ by the substitution $\{X \mapsto \lambda y_1, \dots, y_n. Y(X(y_1, \dots, y_n)\sigma_1, X(y_1, \dots, y_n)\sigma_2)\}$.

Because of top-maximality and shallowness, r_1 and r_2 have genvars in the same positions. The projection-based property implies that r_1 contains only one genvar, which we denoted by Y above. Repeating the above reasoning for each genvar position finishes the proof. ◀

A disadvantage of projection-based generalizations is that if two subterms do not have the same head, projection-based generalization does not focus on their common parts. However, often it is interesting to report the commonalities between such subterms. This is what common-subterm generalization is about.

5.2 Generalization with common subterms

5.2.1 CS-variant

In the definition of cs-generalizations (Definition 10) we just required the set $\{r_1, \dots, r_n\}$ to originate from *some set of common subterms* of $s|_p$ and $t|_p$. Such a relaxed definition will allow us in the next sections to relate the cs-variant to more specific categories such as rfc-, fc-, and pattern variants. However, for the Select function we need a stronger way to choose $\{r_1, \dots, r_n\}$, since we aim at computing lggs. Therefore, we introduce the notion of position-maximal common subterm of two terms:

▶ **Definition 22.** *Let t_1, t_2, and s be terms such that for some positions p_1 of t_1 and p_2 of t_2, we have $t_1|_{p_1} = t_2|_{p_2} = s$. We say that s is a* (p_1, p_2)-maximal common subterm *of t_1 and t_2 if*

- $p_1 = \epsilon$ *or* $p_2 = \epsilon$, *or*
- $p_1 = p'_1.i_1$ *and* $p_2 = p'_2.i_2$ *for some* p'_1, i_1, p'_2, *and* i_2, *and* $t_1|_{p'_1} \neq t_2|_{p'_2}$.

A common subterm of two terms is position-maximal *if it is their (p_1, p_2)-maximal common subterm for some positions p_1 of t_1 and p_2 of t_2.*

The set of position-maximal common subterm occurrence triples of t_1 and t_2 is defined as $pmcso(t_1, t_2) := \{(p_1, p_2, s) \mid s$ is a (p_1, p_2)-maximal common subterm of t_1 and $t_2\}$.

Given an $pmcso(t_1, t_2)$ and a set of variables χ such that no bound variable occurring as the term of a triple of $pmcso(t_1, t_2)$ is in χ, an χ-extension *of $pmcso(t_1, t_2)$ is the set*

$$\begin{aligned} pmcso_\chi(t_1, t_2) &:= pmcso(t_1, t_2) \\ &\cup \{(p, \bullet, x) \mid x \in \chi \setminus \mathsf{fv}(t_2),\ p \text{ is the first position with } t_1|_p = x\} \\ &\cup \{(\bullet, p, x) \mid x \in \chi \setminus \mathsf{fv}(t_1),\ p \text{ is the first position with } t_2|_p = x\}. \end{aligned}$$

Remark. Since it is enough to have one occurrence of $(p, \bullet, x)$ and $(\bullet, p, x)$, it does not matter how p is computed. We can, e.g., assume that it is the first leftmost-outermost position.

▶ **Example 23.** The set of all position-maximal common subterms of $f(g(x), g(x), g(g(x)))$ and $h(g(g(x)), a, b)$ is $\{g(x), g(g(x))\}$, where $g(x)$ is the $(1, 1.1)$- and $(2, 1.1)$-maximal common subterm, and $g(g(x))$ is the $(3, 1)$-maximal common subterm.

Now, we obtain the special case of the Sol rule for position-maximal common subterms by choosing cond and, as a consequence, **QR**, as follows:

Specifying Select$_{\mathsf{cond}}(\{\vec{x}\}, t, s)$: $\mathsf{cond}(\{\vec{x}\}, t, s, Q)$ is true iff $Q = pmcso_{\{\vec{x}\}}(t, s)$.

The instance of QR: $\{q_1, \ldots, q_n\}$ is the $\{\vec{x}\} \cap (\mathsf{fv}(t) \cup \mathsf{fv}(s))$-extension of the set of all position-maximal common subterms of t and s.

cond and the item 2 of the definition of Select (Definition 17) imply that we have $\{\vec{x}\} \cap (\mathsf{fv}(t) \cup \mathsf{fv}(s))$-extension in the instance of **QR**. Without item 2, it would be just $\{\vec{x}\}$-extension. Note that for computing cs-generalizations, if would be sufficient to take $\{\vec{x}\}$-extensions, but we aim at computing cs-lggs, that's why we would keep only the necessary variables from $\{\vec{x}\}$ in generalizations. The necessary ones are those that appear in t or in s.

Yet another remark, which concerns the difference between cs-generalizations and Select is that the q's we get from **QR** form the set of *all position-maximal common subterms* of the terms to be generalized, while in cs-generalization the free variables apply to *some set of common subterms* of those terms. This difference is motivated by our wish to have, on the one hand, rfc-, fc-, and pattern-lggs later as special cs-generalizations and, on the other hand, to compute cs-lggs by the specific version of Sol. The specified instance of cond does not imply any special form of C_t and C_s. They are like it was defined in **CR**.

The obtained instance of Sol is denoted by Sol-CS, and the obtained algorithm by $\mathfrak{G}_{\mathrm{cs}}$. We get the theorem, in which (and in the analogous theorems for rfc, fc, and patterns below) n is the size of the input:

▶ **Theorem 24.** $\mathfrak{G}_{\mathrm{cs}}$ *computes a cs-generalization of two terms in time $O(n^3)$.*

Proof. Top-maximality and genvar-shallowness follow from Theorem 14. The cs-generalization property follows from the instance of **QR**. Selecting position-maximal common subterms from two terms can take quadratic time, and $\mathfrak{G}_{\mathrm{cs}}$ can perform this operation linearly many times. Hence the cubic time complexity. Merging at the end can not make it worse. ◀

▶ **Theorem 25** (Completeness of $\mathfrak{G}_{\text{cs}}$). *Let r_0 be a cs-generalization of t and s. Then $\mathfrak{G}_{\text{cs}}$ computes a generalization r of t and s such that $r_0 \preceq r$.*

Proof sketch. Cs-generalizations differ from each other by the amount of position-maximal common subterms they take in the generalization, and by the number of duplicate generalization variables. The Select function makes sure that $\mathfrak{G}_{\text{cs}}$ puts in generalizations as many position-maximal common subterms as possible, and the exhaustive application of Mer makes all possible sharings of genvars. These arguments imply that $r_0 \preceq r$. ◀

▶ **Corollary 26.** *Cs-variant of higher-order anti-unification is unitary.*

Proof. Follows from Theorem 25 and Theorem 16, since the specification of instances of **QR** and **CR** makes the q's and C's in the Solve rule uniquely determined. ◀

▶ **Example 27.** Let $t = \lambda x_1.f(g_1(x_1,a), g_2(\lambda x_2.h(x_2)))$, $s = \lambda y_1.f(h_1(a,a), h_2(\lambda y_2.h(y_2)))$. $\mathfrak{G}_{\text{cs}}$ gives $\lambda x_1.f(Z_1(x_1,a), Z_2(\lambda x_2.h(x_2)))$. (See Example 44 in Appendix.) If we had $\{\vec{x}\}$-extension in the instance of **QR**, we would get $\lambda x_1.f(Z_1(x_1,a), Z_3(x_1, \lambda x_2.h(x_2)))$, which is more general than $\lambda x_1.f(Z_1(x_1,a), Z_2(\lambda x_2.h(x_2)))$.

▶ **Example 28.** let $t = \lambda x.f(g(x), h(x,a))$ and $s = \lambda y.h(g(y), a)$. Then $\mathfrak{G}_{\text{cs}}$ gives the final state $\emptyset$; $\{Y(y_1,y_2,y_3) : f(y_2, h(y_1,y_3)) \triangleq h(y_2,y_3)\}$; $\lambda x.Y(x, g(x), a)$.

In some applications, it is desirable that the arguments of free variables are not subterms of each other. This requirement leads to generalization for (relaxed) fc- and patterns. These special cases also rely on position-maximal common subterm computation, but the obtained set is filtered. For those variants, in the sections below, we assume that the input terms are closed. Otherwise we will need to add some extra tests to make sure that free variables from the input appear in the generalization only if they do not violate the rfc-, fc-, or patterns restrictions. It will just make things more cumbersome without giving any special insights about the problem. Therefore, for simplicity, we prefer to work with closed input.

For closed input terms, genvar-shallow generalizations are just shallow generalizations. Therefore, below we will mention only the latter.

5.2.2 RFC-variant

From Definition 11 it follows that rfc-generalizations are shallow, but not necessarily top-maximal. Moreover, even top-maximal rfc-generalizations do not have to be cs-generalizations. For instance, if $s = \lambda x.f(h_1(g_1(x)), h_1(g_2(x)))$ and $t = \lambda x.f(h_2(g_1(x)), h_2(g_2(x)))$, then $r = \lambda x.f(X(g_1(x), g_2(x)), X(g_1(x), g_2(x)))$ is an rfc-generalization of s and t, but it is not a cs-generalization. However, top-maximal rfc-lggs are cs-generalizations:

▶ **Theorem 29.** *Let r be a top-maximal rfc-lgg of s and t. Then r is their cs-generalization.*

Proof. Let $X(r_1,\dots,r_n) = r|_p$, where X is a genvar. Since r is an rfc-term, $(\{X\}, \mathsf{B}, X(r_1, \dots, r_n))$ is an rfc-triple, where B is the set of variables bound by λ above the position p. The terms $r_1,\dots,r_n$ should contain all variables from $\mathsf{B} \cap (\mathsf{fv}(s|_p) \cup \mathsf{fv}(t|_p))$, otherwise $X(r_1,\dots,r_n)$ can not generalize $s|_p$ and $t|_p$. Moreover, $r_1,\dots,r_n$ should be common subterms of $s|_p$ and $t|_p$. Otherwise it will violate the assumption that r is an lgg: if, say, r_n is not a common subterm of $s|_p$ and $t|_p$ (in the sense mentioned in the previous section), then $Y(r_1,\dots,r_{n-1})$ will be again a generalization of $s|_p$ and $t|_p$, but less general than $X(r_1,\dots,r_n)$. By the assumption, r is top-maximal. As an rfc-generalization, r is shallow. Since p was an arbitrary position with a genvar, all these conditions imply that r is a cs-generalization of s and t. ◀

Since we aim at computing rfc-lggs, we can take an instance of the Sol rule so that it generates only those rfc-generalizations that are cs-generalizations. We call them {cs, rfc}-generalizations. In them, in addition to the common-subterms condition in Definition 10, the subterms of genvars should satisfy argument and local restrictions. It leads to the instance of Select, in which cond starts from the set Q as in cs-generalizations, and removes from it those terms that violate argument and local restrictions:

Specifying $\mathsf{Select}_{\mathsf{cond}}(\{\vec{x}\}, t, s)$: $\mathsf{cond}(\{\vec{x}\}, t, s, Q)$ is true iff Q is obtained from the set $pmcso_{\{\vec{x}\}}(t, s)$ by removing from it
- **(a)** all triples (p_1, p_2, q) where $q{\downarrow_\eta}$ is not $\{\vec{x}\}$-restricted in $t \triangleq s$[1] and
- **(b)** all triples (p_1^i, p_2^i, q_i) for which there exists $(p_1^j, p_2^j, q_j) \in pmcso_{\{\vec{x}\}}(t, s)$ such that $q_j{\downarrow_\eta} \sqsubset q_i{\downarrow_\eta}$.

The instance of QR: $\{q_1, \ldots, q_n\}$ is the largest set of position-maximal common subterms of t and s whose η-normal forms are $\{\vec{x}\} \cap (\mathsf{fv}(t) \cap \mathsf{fv}(s))$-restricted in t or in s, and none of those η-normal forms are subterms of each other.

Similar to the previous section, the terms C_t and C_s do not have any special form. Defining the q's in this way, it is easy to see that $Y(q_1, \ldots, q_n)$, in the generalization computed by Sol satisfies both the argument restriction and the local restriction. The obtained rule is called Sol-RFC, and the algorithm $\mathfrak{G}_{\mathrm{rfc}}$. We get the theorem:

▶ **Theorem 30.** *$\mathfrak{G}_{\mathrm{rfc}}$ computes a top-maximal {cs, rfc}-generalization in time $O(n^3)$.*

Proof. From Theorem 14 we get top-maximality and shallowness (since the input is assumed to be closed). The {cs, rfc}-property follows from the instance of **QR**. The $O(n^3)$ time of computing cs-generalizations dominates the time needed to filter out subterms that violate the rfc-property (since argument and local restrictions are checked in quadratic time). ◀

▶ **Theorem 31** (Completeness of $\mathfrak{G}_{\mathrm{rfc}}$)**.** *Let r_0 be a top-maximal rfc-generalization of t and s. Then $\mathfrak{G}_{\mathrm{rfc}}$ computes a generalization r of t and s such that $r_0 \preceq r$.*

Proof sketch. Among two top-maximal rfc-generalizations, the one with all position-maximal common subterms and all possible sharings of genvars is less general. ◀

▶ **Corollary 32.** *Rfc-variant of higher-order anti-unification is unitary.*

Proof. Follows from Theorem 31 and Theorem 16, since the specification of instances of **QR** and **CR** makes the q's and C's in the Solve rule uniquely determined. ◀

▶ **Example 33.** Let $t = \lambda x.f(h_1(g(g(x)), a, b), h_2(g(g(x))))$, $s = \lambda y.f(h_3(g(g(y)), g(y), a), h_4(g(g(y))))$. Then $\mathfrak{G}_{\mathrm{rfc}}$ stops with the final state $\emptyset$; $\{Y_1(y_1) : h_1(g(y_1), a, b) \triangleq h_3(g(y_1), y_1, a), Y_2(y_2) : h_2(y_2) \triangleq h_4(y_2)\}$; $\lambda x.f(Y_1(g(x)), Y_2(g(g(x))))$.

5.2.3 FC-variant

Fc-generalizations are also rfc-generalizations and, hence, the properties of rfc-generalizations are valid for fc-generalizations as well. The counterpart of Theorem 29 holds. Analogously to the rfc case, here we aim at computing {cs, fc}-generalizations.

The peculiarity here is that we have to take into account the global condition of fc-terms. Therefore, we need to impose a strategy on the application of the (yet to be defined) instance of the Sol rule: It should be applied only if no other rule applies. Let at this moment A

[1] We look here at $t \triangleq s$ as a term, also in the selection functions for fc- and pattern generalizations later.

be the set $\{X_1(\overrightarrow{x_1}) : t_1 \triangleq s_1, \dots, X_k(\overrightarrow{x_k}) : t_k \triangleq s_k\}$. Let M_i, $1 \leq i \leq m$, be the set of all position-maximal common subterms of t_i and s_i, and let $M = \cup_{i=1}^{k} M_i$. Then we formulate the instance of Select, in which cond takes into account M, and filters out terms violating argument, local, and global restrictions:

Specifying $\mathsf{Select}_{\mathsf{cond}}(\{\vec{x}\}, t, s)$: $\mathsf{cond}(\{\vec{x}\}, t, s, Q)$ is true iff Q is obtained from the set $pmcso_{\{\vec{x}\}}(t, s)$ by

(a) removing all $(p_1, p_2, q) \in pmcso_{\{\vec{x}\}}(t, s)$ where $q{\downarrow_\eta}$ is not $\{\vec{x}\}$-restricted in $t \triangleq s$ and

(b) replacing all $(p_1^i, p_2^i, q_i) \in pmcso_{\{\vec{x}\}}(t, s)$ by (p_1^j, p_2^j, q_j), where $q_j{\downarrow_\eta} \sqsubset q_i{\downarrow_\eta}$ and $q_j \in M$.

Note that the condition **(b)** here includes as a special case the condition **(b)** from the Select instance for rfc-generalizations. This selection function, by Definition 17, leads to the following instance of **QR**:

The instance of QR: Let Q be the largest set of position-maximal common subterms of t and s whose η-normal forms are $\{\vec{x}\} \cap (\mathsf{fv}(t) \cap \mathsf{fv}(s))$-restricted in t or in s. Then $\{q_1, \dots, q_n\}$ is obtained from Q by replacing all $q_i \in Q$ by $q_j \in M$, if $q_j{\downarrow_\eta} \sqsubset q_i{\downarrow_\eta}$.

Since we take into account the whole of M when deciding which subterms to keep under the genvars, the global restriction of fc-terms is satisfied. Similar to the cs- and rfc-variants, the terms C_t and C_s here do not have any special form. The obtained instance of Sol is denoted by Sol-FC, and the algorithm by $\mathfrak{G}_{\mathrm{fc}}$. The theorems below can be proved similarly to their rfc-counterparts:

▶ **Theorem 34.** *$\mathfrak{G}_{\mathrm{fc}}$ computes a top-maximal {cs, fc}-generalization in time $O(n^3)$.*

▶ **Theorem 35** (Completeness of $\mathfrak{G}_{\mathrm{fc}}$)**.** *Let r_0 be a top-maximal fc-generalization of t and s. Then $\mathfrak{G}_{\mathrm{fc}}$ computes a generalization r of t and s such that $r_0 \preceq r$.*

▶ **Corollary 36.** *Fc-variant of higher-order anti-unification is unitary.*

▶ **Example 37.** For terms in Example 33, $\mathfrak{G}_{\mathrm{fc}}$ stops with the final state $\emptyset$; $\{Y_1(y_1) : h_1(g(y_1), a, b) \triangleq h_3(g(y_1), y_1, a),\ Y_2(y_2) : h_2(g(y_2)) \triangleq h_4(g(y_2))\}$; $\lambda x.f(Y_1(g(x)), Y_2(g(x)))$.

5.2.4 Pattern variant

Similarly to rfc- and fc-generalizations, pattern generalizations are shallow but not necessarily top-maximal (and, consequently, not cs-generalizations). For instance, $\lambda x, y.f(X(x, y))$ is a pattern generalization of $s = t = \lambda x, y.f(g(x))$, which is neither top-maximal nor cs-generalization. However, pattern lggs are top-maximal and retain common subterms (note the difference from rfc- and fc-generalization, where lggs are not necessarily top-maximal):

▶ **Theorem 38.** *A least general pattern generalization of two terms is their cs-generalization.*

Proof. Top-maximality of pattern lgg follows from completeness of pattern generalization algorithm described in [7, 8]. The rest of the proof is similar to the proof of Theorem 29. ◀

Specifying $\mathsf{Select}_{\mathsf{cond}}(\{\vec{x}\}, t, s)$: $\mathsf{cond}(\{\vec{x}\}, t, s, Q)$ is true iff Q is obtained from the set $pmcso_{\{\vec{x}\}}(t, s)$ by

(a) removing all $(p_1, p_2, q) \in pmcso_{\{\vec{x}\}}(t, s)$ where $q{\downarrow_\eta}$ is not $\{\vec{x}\}$-restricted in $t \triangleq s$ and

(b) replacing all $(p_1^i, p_2^i, q_i) \in pmcso_{\{\vec{x}\}}(t, s)$ by (p_1^j, p_2^j, x), where $x{\downarrow_\eta} \sqsubset q_i{\downarrow_\eta}$ and $x \in \{\vec{x}\}$.

We wrote Select in this form to relate it to the selection functions of the other common subterms based generalizations (cs, rfc, fc). It leads to the instances of **QR** and **CR**:

The instance of QR: $\{q_1, \ldots, q_n\} = \{\vec{x}\} \cap (\mathsf{fv}(t) \cup \mathsf{fv}(s))$.
The instance of CR: $C_t = t$, $C_s = s$, $y_i = q_i$.

The obtained instance of Sol is denoted by Sol-P, and the obtained algorithm by $\mathfrak{G}_{\text{pat}}$. It is, in fact, the algorithm from [8], for the closed input. It is complete. The theorem below is also from [8]:

▶ **Theorem 39.** $\mathfrak{G}_{\text{pat}}$ *computes a least general pattern generalization in time* $O(n)$.

It is known from [8] that pattern variant of higher-order anti-unification is unitary. It can be also seen from Theorem 16 and the definitions of the instances of **QR** and **CR** above, which makes the choice of the q's and C's in Sol unique.

6 Conclusion

We described a general framework for computing top-maximal genvar-shallow generalizations of two terms and proved its properties. Appropriate instantiation of the framework gives concrete instances of variants of higher-order anti-unification. By instantiations, we obtained four new unitary variants of higher-order generalization.

References

1 Hassan Aït-Kaci and Gabriella Pasi. Fuzzy Unification and Generalization of First-Order Terms over Similar Signatures. In Fabio Fioravanti and John P. Gallagher, editors, *LOPSTR 2017*, volume 10855 of *LNCS*, pages 218–234. Springer, 2017. doi:10.1007/978-3-319-94460-9_13.
2 María Alpuente, Demis Ballis, Angel Cuenca-Ortega, Santiago Escobar, and José Meseguer. ACUOS2: A high-performance system for modular ACU generalization with subtyping and inheritance. In *16th European Conference on Logics in Artificial Intelligence, JELIA 2019*, LNCS. Springer, 2019. To appear.
3 María Alpuente, Santiago Escobar, Javier Espert, and José Meseguer. A modular order-sorted equational generalization algorithm. *Inf. Comput.*, 235:98–136, 2014. doi:10.1016/j.ic.2014.01.006.
4 Hendrik Pieter Barendregt, Wil Dekkers, and Richard Statman. *Lambda Calculus with Types*. Perspectives in logic. Cambridge University Press, 2013.
5 Adam D. Barwell, Christopher Brown, and Kevin Hammond. Finding parallel functional pearls: Automatic parallel recursion scheme detection in Haskell functions via anti-unification. *Future Generation Comp. Syst.*, 79:669–686, 2018. doi:10.1016/j.future.2017.07.024.
6 Alexander Baumgartner and Temur Kutsia. Unranked second-order anti-unification. *Inf. Comput.*, 255:262–286, 2017. doi:10.1016/j.ic.2017.01.005.
7 Alexander Baumgartner, Temur Kutsia, Jordi Levy, and Mateu Villaret. A Variant of Higher-Order Anti-Unification. In Femke van Raamsdonk, editor, *24th International Conference on Rewriting Techniques and Applications, RTA 2013, June 24-26, 2013, Eindhoven, The Netherlands*, volume 21 of *LIPIcs*, pages 113–127. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013. doi:10.4230/LIPIcs.RTA.2013.113.
8 Alexander Baumgartner, Temur Kutsia, Jordi Levy, and Mateu Villaret. Higher-Order Pattern Anti-Unification in Linear Time. *J. Autom. Reasoning*, 58(2):293–310, 2017. doi:10.1007/s10817-016-9383-3.
9 Cao Feng and Stephen Muggleton. Towards Inductive Generalization in Higher Order Logic. In Derek H. Sleeman and Peter Edwards, editors, *Proceedings of the Ninth International Workshop on Machine Learning (ML 1992), Aberdeen, Scotland, UK, July 1-3, 1992*, pages 154–162. Morgan Kaufmann, 1992.

10 Kouichi Hirata, Takeshi Ogawa, and Masateru Harao. Generalization Algorithms for Second-Order Terms. In Rui Camacho, Ross D. King, and Ashwin Srinivasan, editors, *Inductive Logic Programming, 14th International Conference, ILP 2004, Porto, Portugal, September 6-8, 2004, Proceedings*, volume 3194 of *Lecture Notes in Computer Science*, pages 147–163. Springer, 2004. `doi:10.1007/978-3-540-30109-7_14`.

11 Temur Kutsia, Jordi Levy, and Mateu Villaret. Anti-unification for Unranked Terms and Hedges. *J. Autom. Reasoning*, 52(2):155–190, 2014. `doi:10.1007/s10817-013-9285-6`.

12 Tomer Libal and Dale Miller. Functions-as-Constructors Higher-Order Unification. In Delia Kesner and Brigitte Pientka, editors, *1st International Conference on Formal Structures for Computation and Deduction, FSCD 2016, June 22-26, 2016, Porto, Portugal*, volume 52 of *LIPIcs*, pages 26:1–26:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. `doi:10.4230/LIPIcs.FSCD.2016.26`.

13 José Meseguer. Symbolic Reasoning Methods in Rewriting Logic and Maude. In Lawrence S. Moss, Ruy J. G. B. de Queiroz, and Maricarmen Martínez, editors, *WoLLIC 2018*, volume 10944 of *LNCS*, pages 25–60. Springer, 2018. `doi:10.1007/978-3-662-57669-4_2`.

14 Dale Miller. A Logic Programming Language with Lambda-Abstraction, Function Variables, and Simple Unification. *J. Log. Comput.*, 1(4):497–536, 1991. `doi:10.1093/logcom/1.4.497`.

15 Frank Pfenning. Unification and Anti-Unification in the Calculus of Constructions. In *Proceedings of the Sixth Annual Symposium on Logic in Computer Science (LICS '91), Amsterdam, The Netherlands, July 15-18, 1991*, pages 74–85. IEEE Computer Society, 1991. `doi:10.1109/LICS.1991.151632`.

16 Brigitte Pientka. Higher-order term indexing using substitution trees. *ACM Trans. Comput. Log.*, 11(1), 2009. `doi:10.1145/1614431.1614437`.

17 Gordon D. Plotkin. A note on inductive generalization. *Machine Intell.*, 5(1):153–163, 1970.

18 John C. Reynolds. Transformational systems and the algebraic structure of atomic formulas. *Machine Intell.*, 5(1):135–151, 1970.

19 Reudismam Rolim, Gustavo Soares, Rohit Gheyi, and Loris D'Antoni. Learning Quick Fixes from Code Repositories. *CoRR*, abs/1803.03806, 2018. `arXiv:1803.03806`.

20 Reudismam Rolim de Sousa. *Learning syntactic program transformations from examples.* PhD thesis, Universidade Federal de Campina Grande, Brazil, 2018.

21 Martin Schmidt, Ulf Krumnack, Helmar Gust, and Kai-Uwe Kühnberger. Heuristic-Driven Theory Projection: An Overview. In Henri Prade and Gilles Richard, editors, *Computational Approaches to Analogical Reasoning: Current Trends*, volume 548 of *Studies in Computational Intelligence*, pages 163–194. Springer, 2014. `doi:10.1007/978-3-642-54516-0_7`.

A Examples

▶ **Example 40.** Let $t = \lambda x.f(g(x), g(g(x)))$ and $s = \lambda x.h(g(g(x)), g(x))$. Then

- $r_0 = \lambda x.X(f(g(x), g(g(x))), h(g(g(x)), g(x)))$ is a shallow top-maximal lgg of t and s.
- $r_1 = \lambda x.Y(g(x), g(g(x)))$ is a top-maximal rfc-lgg of t and s. We have $r_1 \prec r_0$. Note that $r_2 = \lambda x.Y(g(g(x)), g(x))$ is also a top-maximal rfc-generalization with $r_1 \simeq r_2$.
- $r_3 = \lambda x.Z(g(x))$ is a top-maximal fc-lgg of t and s. In this case we have $r_3 \simeq r_1$, because $r_3\{Z \mapsto \lambda x.Y(x, g(x))\} = r_1$ and $r_1\{Y \mapsto \lambda x, y.Z(x)\} = r_3$.
- $r_4 = \lambda x.X(x)$ is a top-maximal pattern-lgg of t and s and $r_4 \prec r_3$.

▶ **Example 41.** Let $t = \lambda x.f(x, x)$ and $s = \lambda x.f(g(g(x)), g(x))$. Then the non-shallow term $r = \lambda x.f(Y(Y(x)), Y(x))$ is a top-maximal generalization of t and s, and it is less general than their shallow top-maximal generalization $\lambda x.f(Z(x, g(g(x))), Z(x, g(x)))$.

▶ **Example 42.** Let $t = \lambda x, y.X(f(x), f(y))$ and $s = \lambda x, y.X(g(y), g(x))$. Then the term $r = \lambda x, y.X(Y(f(x), g(y)), Y(f(y), g(x)))$ is a $\mathsf{genvar}(r, t, s)$-shallow top-maximal lgg of t and s, but not a shallow top-maximal lgg.

▶ **Example 43.** Let $t = \lambda x.f(g(x), g(x), g(g(x)))$ and $s = \lambda y.h(g(g(y)), a, b)$. Then the sequence of inferences in $\mathfrak{G}_{\text{cs}}$ is

$$\{X : \lambda x.f(g(x), g(x), g(g(x))) \triangleq \lambda y.h(g(g(y)), a, b)\};\ \emptyset;\ X \Longrightarrow_{\mathsf{Abs}}$$
$$\{X'(x) : f(g(x), g(x), g(g(x))) \triangleq h(g(g(x)), a, b)\};\ \emptyset;\ \lambda x.X'(x) \Longrightarrow_{\mathsf{Sol\text{-}CS}}$$
$$\emptyset;\ \{Y(y_1, y_2) : f(y_1, y_1, y_2) \triangleq h(y_2, a, b)\};\ \lambda x.Y(g(x), g(g(x))).$$

In the Sol-CS step, we compute all position-maximal common subterms and their positions as in Example 23. Therefore, $f(y_1, y_1, y_2)$ and $h(y_2, a, b)$ are obtained from

$$f(g(x), g(x), g(g(x)))[1 \mapsto y_1][2 \mapsto y_1][3 \mapsto y_2] \text{ and}$$
$$h(g(g(x)), a, b)[1.1 \mapsto y_1][1.1 \mapsto y_1][1 \mapsto y_2],$$

respectively. To obtain t (resp., s) from the computed generalization $\lambda x.Y(g(x), g(g(x)))$, we need to apply the substitution $\{Y \mapsto \lambda y_1, y_2.f(y_1, y_1, y_2)\}$ (resp., $\{Y \mapsto \lambda y_1, y_2.h(y_2, a, b)\}$) to it. These substitutions can be directly read off the store.

▶ **Example 44.** Let $t = \lambda x_1.f(g_1(x_1, a), g_2(\lambda x_2.h(x_2)))$, $s = \lambda y_1.f(h_1(a, a), h_2(\lambda y_2.h(y_2)))$. Then we get the following derivation in $\mathfrak{G}_{\text{cs}}$:

$$\{X : \lambda x_1.f(g_1(x_1, a), g_2(\lambda x_2.h(x_2))) \triangleq \lambda y_1.f(h_1(a, a), h_2(\lambda y_2.h(y_2)))\};\ \emptyset;\ X \Longrightarrow_{\mathsf{Abs}}$$
$$\{Y(x_1) : f(g_1(x_1, a), g_2(\lambda x_3.h(x_2))) \triangleq f(h_1(a, a), h_2(\lambda y_2.h(y_2)))\};\ \emptyset;\ \lambda x_1.Y(x_1) \Longrightarrow_{\mathsf{Dec}}$$
$$\{Y_1(x_1) : g_1(x_1, a) \triangleq h_1(a, a),\ Y_2(x_1) : g_2(\lambda x_2.h(x_2)) \triangleq h_2(\lambda y_2.h(y_2))\};\ \emptyset;$$
$$\lambda x_1.f(Y_1(x_1), Y_2(x_1))$$

Here Sol-CS rule applies. The set of position-maximal common subterms of $g_1(x_1, a)$ and $h_1(a, a)$ is $\{a\}$. We need to extend it by x_1, because x_1 has been bound before (as $Y_1(x_1)$ tells) and it appears in $g_1(x_1, a)$. Hence, after this extension we get the set $\{x_1, a\}$, which will be introduced in the generalization. The store also changes correspondingly:

$$\{Y_2(x_1) : g_2(\lambda x_2.h(x_2)) \triangleq h_2(\lambda y_2.h(y_2))\};\ \{Z_1(z_1, z_2) : g_1(z_1, z_2) \triangleq h_1(z_2, z_2)\};$$
$$\lambda x_1.f(Z_1(x_1, a), Y_2(x_1))$$

Also here, we use Sol-CS. The set of position-maximal common subterms of $g_2(\lambda x_2.h(x_2))$ and $h_2(\lambda y_2.h(y_2))$ is $\{\lambda x_2.h(x_2)\}$ (modulo α-equivalence). This set will not be extended by any bound variable, because the only candidate, x_1, appears neither in $g_2(\lambda x_2.h(x_2))$ nor in $h_2(\lambda y_2.h(y_2))$. Therefore, we get

$$\emptyset;\ \{Z_1(z_1, z_2) : g_1(z_1, z_2) \triangleq h_1(z_2, z_2),\ Z_2(z_3) : g_2(z_3) \triangleq h_2(z_3)\};$$
$$\lambda x_1.f(Z_1(x_1, a), Z_2(\lambda x_2.h(x_2))).$$

Note that if we had $\{\vec{x}\}$-extension instead of $\{\vec{x}\} \cap (\mathsf{fv}(t) \cup \mathsf{fv}(s))$-extension in the instance of **QR** above, then in the last step we would get the generalization $\lambda x_1.f(Z_1(x_1, a), Z_2(x_1, \lambda x_2.h(x_2)))$, which is more general than $\lambda x_1.f(Z_1(x_1, a), Z_2(\lambda x_2.h(x_2)))$, computed by $\mathfrak{G}_{\text{cs}}$.

▶ **Example 45.** Let t and s be the terms, $t = \lambda x.f(h_1(g(g(x)), a, b), h_2(g(g(x))))$, $s = \lambda y.f(h_3(g(g(y)), g(y), a), h_4(g(g(y))))$. Then $\mathfrak{G}_{\text{rfc}}$ performs the following steps:

$$\{X : \lambda x.f(h_1(g(g(x)), a, b), h_2(g(g(x)))) \triangleq$$
$$\lambda y.f(h_3(g(g(y)), g(y), a), h_4(g(g(y))))\};\ \emptyset;\ X \Longrightarrow_{\mathsf{Abs}}$$

$$\{X'(x) : f(h_1(g(g(x)), a, b), h_2(g(g(x)))) \triangleq$$
$$f(h_3(g(g(x)), g(x), a), h_4(g(g(x))))\}; \emptyset;\ \lambda x.X'(x) \Longrightarrow_{\mathsf{Dec}}$$
$$\{Z_1(x) : h_1(g(g(x)), a, b) \triangleq h_3(g(g(x)), g(x), a),$$
$$Z_2(x) : h_2(g(g(x))) \triangleq h_4(g(g(x)))\};\ \emptyset;\ \lambda x.f(Z_1(x), Z_2(x)) \Longrightarrow_{\mathsf{Sol\text{-}RFC}}$$
$$\{Z_2(x) : h_2(g(g(x))) \triangleq h_4(g(g(x)))\};$$
$$\{Y_1(y_1) : h_1(g(y_1), a, b) \triangleq h_3(g(y_1), y_1, a)\};\ \lambda x.f(Y_1(g(x)), Z_2(x)) \Longrightarrow_{\mathsf{Sol\text{-}RFC}}$$
$$\emptyset;\ \{Y_1(y_1) : h_1(g(y_1), a, b) \triangleq h_3(g(y_1), y_1, a),\ Y_2(y_2) : h_2(y_2) \triangleq h_4(y_2)\};$$
$$\lambda x.f(Y_1(g(x)), Y_2(g(g(x)))).$$

► **Example 46.** Let us see how fc-generalization can be computed for terms in Example 45. We can show the part of the computation that starts with Sol-FC:

$$\{Z_1(x) : h_1(g(g(x)), a, b) \triangleq h_3(g(g(x)), g(x), a),$$
$$Z_2(x) : h_2(g(g(x))) \triangleq h_4(g(g(x)))\};\ \emptyset;\ \lambda x.f(Z_1(x), Z_2(x)) \Longrightarrow_{\mathsf{Sol\text{-}FC}}$$
$$\{Z_2(x) : h_2(g(g(x))) \triangleq h_4(g(g(x)))\};$$
$$\{Y_1(y_1) : h_1(g(y_1), a, b) \triangleq h_3(g(y_1), y_1, a)\}; \lambda x.f(Y_1(g(x)), Z_2(x)) \Longrightarrow_{\mathsf{Sol\text{-}FC}}$$
$$\emptyset;\ \{Y_1(y_1) : h_1(g(y_1), a, b) \triangleq h_3(g(y_1), y_1, a),\ Y_2(y_2) : h_2(g(y_2)) \triangleq h_4(g(y_2))\};$$
$$\lambda x.f(Y_1(g(x)), Y_2(g(x))).$$

Homotopy Canonicity for Cubical Type Theory

Thierry Coquand
Department of Computer Science and Engineering, University of Gothenburg, Sweden
coquand@chalmers.se

Simon Huber
Department of Computer Science and Engineering, University of Gothenburg, Sweden
simon.huber@cse.gu.se

Christian Sattler
Department of Computer Science and Engineering, University of Gothenburg, Sweden
sattler@chalmers.se

Abstract

Cubical type theory provides a constructive justification of homotopy type theory and satisfies canonicity: every natural number is convertible to a numeral. A crucial ingredient of cubical type theory is a path lifting operation which is explained computationally by induction on the type involving several non-canonical choices. In this paper we show by a sconing argument that if we remove these equations for the path lifting operation from the system, we still retain *homotopy* canonicity: every natural number is *path equal* to a numeral.

2012 ACM Subject Classification Theory of computation → Type theory; Theory of computation → Proof theory; Theory of computation → Computability

Keywords and phrases cubical type theory, univalence, canonicity, sconing, Artin glueing

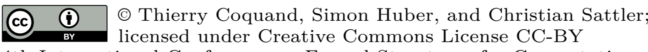
Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.11

Funding *Simon Huber*: I acknowledge the support of the Centre for Advanced Study (CAS) in Oslo, Norway, which funded and hosted the research project Homotopy Type Theory and Univalent Foundations during the academic year 2018/19.

Introduction

This paper is a contribution to the analysis of the computational content of the univalence axiom [34] (and higher inductive types). In previous work [2, 4, 6, 7, 23], various *presheaf models* of this axiom have been described in a constructive meta theory. In this formalism, the notion of fibrant type is stated as a refinement of the path lifting operation where one not only provides one of the endpoints but also a partial lift (for a suitable notion of partiality). This generalized form of path lifting operation is a way to state a homotopy extension property, which was recognized very early (see, e.g. [10]) as a key for an abstract development of algebraic topology. The axiom of univalence is then captured by a suitable *equivalence extension operation* (the "glueing" operation), which expresses that we can extend a partially defined equivalence of a given total codomain to a total equivalence. These presheaf models suggest possible extensions of type theory where we manipulate higher dimensional objects [2, 6]. One can define a notion of reduction and prove canonicity for this extension [16]: any closed term of type $\mathtt{N}$ (natural number) is convertible to a numeral. There are however several *non-canonical* choices when defining the path lifting operation by induction on the type, which produce different notion of convertibility.[1] A natural question is how *essential* these non-canonical choices are: can it be that a closed term of type $\mathtt{N}$, defined without use of such non-canonical reduction rules, becomes convertible to 0 for one choice and 1 for another?

[1] For instance, the definition of this operation for "glue" types is different in [6] and [23].


© Thierry Coquand, Simon Huber, and Christian Sattler;
licensed under Creative Commons License CC-BY
4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019).
Editor: Herman Geuvers; Article No. 11; pp. 11:1–11:23
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

$\mathtt{\Pi}(A,B)'(w)$	$=$	$\Pi(u:\lvert A\rvert)\Pi(u':A'u).B'uu'(\mathtt{app}(w,u))$
$\mathtt{\Sigma}(A,B)'(w)$	$=$	$\Sigma(u':A'(\mathtt{fst}(w))).B'(\mathtt{fst}(w))\,u'\,(\mathtt{snd}(w))$
$\mathtt{fill}_{\psi,b}(u)'$	$=$	$\mathsf{fill}_{\psi,b}(u')$
$\mathtt{Path}(A,a_0,a_1)'(w)$	$=$	$\mathsf{Path}_{\lambda i.A'\,i\,(\mathtt{ap}(w,i))}\,a_0'\,a_1'$
$\mathtt{Glue}(A,\psi\mapsto(B,w))'v$	$=$	$\mathsf{Glue}\;(A'(\mathtt{app}(\mathtt{unglue},v)))\;[\psi\mapsto(B'v,(w'.1\,v,\ldots))]$

Figure 1 The main rules for the sconing model.

The main result of this paper, the *homotopy canonicity theorem*, implies that this cannot be the case. the value of a term is independent of these non-canonical choices. Homotopy canoncity states that, even without providing reduction rules for path lifting operations at type formers, we *still* have that any closed term of type $\mathtt{N}$ is *path equal* to a numeral. (We cannot hope to have convertibility anymore with these path lifting constant.) We can then see this numeral as the "value" of the given term.

Our proof of the homotopy canonicity can be seen as a proof-relevant extension of the *reducibility* or *computability* method, going back to the work of Gödel [14] and Tait [32]. It is however best expressed in an *algebraic* setting. We first define a general notion of model, called *cubical category with families*, defined as a category with families [9] with certain special operations internal to presheaves over a category $\mathcal{C}$ (such as a cube category) with respect to the parameters of an *interval* $\mathbb{I}$ and an object of *cofibrant propositions* $\mathbb{F}$.

We describe the term model and how to re-interpret the cubical presheaf models as cubical categories with families. The computability method can then be expressed as a general operation (called "sconing") which applied to an arbitrary model $\mathcal{M}$ produces a new model $\mathcal{M}^*$ with a strict morphism $\mathcal{M}^* \to \mathcal{M}$. Homotopy canonicity is obtained by applying this general operation to the initial model, which we conjecture to be the term model. This construction associates to a (for simplicity, closed) type A a predicate A' on the closed terms $|A|$, and each closed term u a proof u' of $A'u$. The main rules in the closed case are summarized in Figure 1.

Some extensions and variations are then described:

- Our development extends uniformly to identity types and higher inductive types (using the methods of [7]) (Sections 5.1 and 5.2).
- Our development applies equally to the case where one treats univalence instead of glue types as primitive (Appendix C.1). We expect that a similar sconing argument (glueing along a global sections functor to simplicial sets) works to establish homotopy canonicity for the initial split univalent simplicial tribe in the setting of Joyal [17].
- A similar sconing argument adapts to canonicity of cubical type theory with computation of filling at type formers, originally proved by [16] (Appendix C.2).
- Assuming excluded middle, a version of the simplicial set model [18] forms an instance of our development, and distributive lattice cubical type theory interprets in it (Appendix D).

Using our technique, one may also reprove canonicity for ordinary Martin-Löf type theory with inductive families in a reduction-free way.

Shulman [27] proves homotopy canonicity for homotopy type theory with a truncatedness assumption using the sconing technique. This proof was one starting point for the present work. Some of his constructions may be simplified using our techniques, for example the construction of the natural number type in the sconing.

Parametricity interpretation

As pointed out in [8], there is a strong analogy between proving canonicity via (Artin) glueing and parametricity. For readers more familiar with parametricity than the general (Artin) glueing technique, we motivate our proof by briefly presenting a parametricity interpretation [3] of cubical type theory [6]. The parametricity interpretation is a purely syntactical interpretation which associates by structural induction a family $A'\,x$ where $x : A$ to any type A and a term $t' : A'\,t$ to any term $t : A$. In general, if A is defined over the context $x_1 : A_1, \ldots, x_n : A_n$, then $A'(x_1, x_1, \ldots, x_n, x'_n)$ will be defined over the context $x_1 : A_1, x'_1 : A'_1\,x_1, x_2 : A_2(x_1), x'_2 : A'_2(x_1, x'_1)\,x'_2, \ldots$. The interpretation of Π, Σ, and universes is the same as for ordinary type theory. For instance,

$$(\Pi(x : A)B)'\,w = \Pi(x : A)(x' : A'x).B'(x, x')\,(w\,x)$$

with $(\lambda(x : A).t)' = \lambda(x : A)(x' : A'x).t'$ and $(t\,u)' = t'\,u\,u'$.

For inductive types, we use the "inductive-style" interpretation [3], so that for instance $N'\,x$ is the inductive family with constructors $0'$ of type $N'\,0$ and S' of type $\Pi(x : N).N'x \to N'\,(S\,x)$. (The other "deductive-style" presentation will not work for the canonicity proof.)

This interpretation works as well for cubical type theory. There is a natural interpretation of path types: $(\mathsf{Path}\,A\,a_0\,a_1)'\,\omega$ is $\mathsf{Path}^i\,(A'\,(\omega\ i))\,a'_0\,a'_1$ given a'_0 in $A'\,a_0$ and a'_1 in $A'\,a_1$. We also take $(\langle i\rangle\,t)' = \langle i\rangle\,t'$ and $(t\,r)' = t'\,r$ for $r : \mathbb{I}$.

Consider $T = \mathsf{Glue}\,A\,[\psi \mapsto (B, w)]$ where A is a "total" type and w a "partial" equivalence between B and A of extent ψ, so a pair of $w.1 : B \to A$ and $w.2$ proving that $w.1$ is an equivalence. We define $T'\,u = \mathsf{Glue}\,A'\,(\mathsf{unglue}\,u)\,[\psi \mapsto (B'\,u, (w.1'\,u, c\,u)]$. We have $\mathsf{unglue} : T \to A$ and we can then define $\mathsf{unglue}' : \Pi(u : T).T'\,u \to A'\,(\mathsf{unglue}\ u)$ by $\mathsf{unglue}'\,u\,u' = \mathsf{unglue}\ u'$. For this, we need to build a proof $c\,u$ that $w.1'\ u$ is an equivalence. This is possible by showing that the map $\Sigma(y : B)B'y \to \Sigma(x : A)A'\,x$ sending (u, u') to $(w.1\,u, w.1'\,u\,u')$ is an equivalence. We can then use Theorem 4.7.7 of [33] about equivalences on total spaces.

There is a problem however at this point: this interpretation does not need to validate the computation rules for the filling operation of the "Glue" type. We can notice however that this problem is solved (in a somewhat trivial way) if we consider a system where the filling operation is given as a *primitive constant*, without any computation rule.

One way to understand the parametricity interpretation is that it is (Artin) glueing of the syntactic model along the identity map. For proving canonicity, we instead glue the initial model with the cubical set model along the global section functor. As in [8], we think that this interpretation is best described in an *algebraic* way, using what is essentially a generalized algebraic presentation of type theory.

Setting

We work in a constructive set theory (as presented e.g. in [1]) with a sufficiently long cumulative hierarchy of Grothendieck universes. However, our constructions are not specific to this setting and can be replayed in other constructive metatheories such as extensional type theory. In Appendix D, we assume classical logic for the discussion of models in simplicial sets.

1 Cubical categories with families

We first recall the notion of categories with families (cwf) [9] equipped with Π- and Σ-types, universes and natural number types. This notion can be interpreted in any presheaf model. In a presheaf model however, we can consider new operations. A *cubical cwf* will be such a cwf in a presheaf model with extra operations which make use of an interval presheaf $\mathbb{I}$ and a type of cofibrant propositions $\mathbb{F}$ as introduced in [7, 23].

1.1 Category with families

Categories with families form an algebraic notion of model of type theory. In order to later model universes à la Russell, we define them in a stratified manner where instead of a single presheaf of types, we specify a filtration of presheaves of "small" types.[2] The length of the filtration is not essential: we have chosen $1+\omega$ so that we may specify constructions just at the top level.

A *category with families* (cwf) consists of the following data.

- We have a category of *contexts* Con and *substitutions* $\mathsf{Hom}(\Delta,\Gamma)$ from Δ to Γ in Con. The identity substitution on Γ in Con is written id, and the composition of δ in $\mathsf{Hom}(\Theta,\Delta)$ and σ in $\mathsf{Hom}(\Delta,\Gamma)$ is written $\sigma\delta$.
- We have a presheaf Type of *types* over the category of contexts. The action of σ in $\mathsf{Hom}(\Delta,\Gamma)$ on a type A over Γ is written $A\sigma$. We have a cumulative sequence of subpresheaves Type_n of *types of level* n of Type where n is a natural number.
- We have a presheaf Elem of *elements* over the category of elements of Type, i.e. a type $\mathsf{Elem}(\Gamma,A)$ for A in $\mathsf{Type}(\Gamma)$ with $a\sigma$ in $\mathsf{Elem}(\Delta,A\sigma)$ for a in $\mathsf{Elem}(\Gamma,A)$ and σ in $\mathsf{Hom}(\Delta,\Gamma)$ satisfying evident laws.
- We have a terminal context 1, with the unique element of $\mathsf{Hom}(\Gamma,1)$ written $()$.
- Given A in $\mathsf{Type}(\Gamma)$, we have a *context extension* $\Gamma.A$. There is a *projection* p in $\mathsf{Hom}(\Gamma.A,\Gamma)$ and a *generic term* q in $\mathsf{Elem}(\Gamma.A,A\mathsf{p})$. Given σ in $\mathsf{Hom}(\Delta,\Gamma)$, A in $\mathsf{Type}(\Gamma)$, and a in $\mathsf{Elem}(\Delta,A\sigma)$ we have a *substitution extension* (σ,a) in $\mathsf{Hom}(\Delta,\Gamma.A)$. These operations satisfy $\mathsf{p}(\sigma,a)=\sigma$, $\mathsf{q}(\sigma,a)=a$, and $(\mathsf{p}\sigma,\mathsf{q}\sigma)=\sigma$. Thus, every element of $\mathsf{Hom}(\Delta,\Gamma.A)$ is uniquely of the form (σ,a) with σ and a as above.

We introduce some shorthand notation related to substitution. Given σ in $\mathsf{Hom}(\Delta,\Gamma)$ and A in $\mathsf{Type}(\Gamma)$, we write $\sigma^+=(\sigma\mathsf{p},\mathsf{q})$ in $\mathsf{Hom}(\Delta.A\sigma,\Gamma.A)$. Given a in $\mathsf{Elem}(\Gamma,A)$, we write $[a]=(\mathsf{id},a)$ in $\mathsf{Hom}(\Gamma,\Gamma.A)$. Thus, given B in $\mathsf{Type}_n(\Gamma.A)$ and a in $\mathsf{Elem}(\Gamma,A)$, we have $B[a]$ in $\mathsf{Type}(\Gamma)$. Given furthermore b in $\mathsf{Elem}(\Gamma.A,B)$, we have $b[a]$ in $\mathsf{Elem}(\Gamma,B[a])$. We extend this notation to several arguments: given a_i in $\mathsf{Elem}(\Gamma,A_i)$ for $1\le i\le k$, we write $[a_1,\ldots,a_k]$ for $[a_k][a_{k-1}\mathsf{p}]\cdots[a_1\mathsf{p}\ldots\mathsf{p}]$ in $\mathsf{Hom}(\Gamma,\Gamma.A_1.\ldots.A_k)$.

Given a cwf as above, we define what it means to have the following type formers. In addition to the specified laws, all specified operations are furthermore required to be stable under substitution in the evident manner.

- **Dependent products.** For A in $\mathsf{Type}(\Gamma)$ and B in $\mathsf{Type}(\Gamma.A)$, we have $\Pi(A,B)$ in $\mathsf{Type}(\Gamma)$, of level n if A and B are. Given b in $\mathsf{Elem}(\Gamma.A,B)$, we have the *abstraction* $\lambda(b)$ in $\mathsf{Elem}(\Gamma,\Pi(A,B))$. Given c in $\mathsf{Elem}(\Gamma,\Pi(A,B))$ and a in $\mathsf{Elem}(\Gamma,A)$, we have the *application* $\mathtt{app}(c,a)$ in $\mathsf{Elem}(\Gamma,B[a])$. These operations satisfy $\mathtt{app}(\lambda(b),a)=b[a]$ and $\lambda(\mathtt{app}(c\mathsf{p},\mathsf{q}))=c$.
  Given A and B in $\mathsf{Type}(\Gamma)$ we write $A\to B$ for $\Pi(A,B\mathsf{p})$.
- **Dependent sums.** For A in $\mathsf{Type}(\Gamma)$ and B in $\mathsf{Type}(\Gamma.A)$, we have $\Sigma(A,B)$ in $\mathsf{Type}(\Gamma)$, of level n if A and B are. Given a in $\mathsf{Elem}(\Gamma,A)$ and b in $\mathsf{Elem}(\Gamma,B[a])$, we have the *pairing* $\mathtt{pair}(a,b)$ in $\mathsf{Elem}(\Gamma,\Sigma(A,B))$. Given c in $\mathsf{Elem}(\Gamma,\Sigma(A,B))$, we have the *first projection* $\mathtt{fst}(c)$ in $\mathsf{Elem}(\Gamma,A)$ and *second projection* $\mathtt{snd}(c)$ in $\mathsf{Elem}(\Gamma,B[\mathtt{fst}(c)])$. These operations satisfy $\mathtt{fst}(\mathtt{pair}(a,b))=a$, $\mathtt{snd}(\mathtt{pair}(a,b))=b$, and $\mathtt{pair}(\mathtt{fst}(c),\mathtt{snd}(c))=c$. Thus, every element of $\mathsf{Elem}(\Gamma,\Sigma(A,B))$ is uniquely of the form $\mathtt{pair}(a,b)$ with a and b as above.

---

[2] We note that this non-algebraic aspect of the definition does not interfere with the otherwise algebraic character. Subset inclusions and equalities of sets $\mathsf{Elem}(\Gamma,\mathtt{U}_n)=\mathsf{Type}_n$ could in principle be replaced by injections and natural isomorphisms, respectively. Then our cwfs become models of a generalized algebraic theory without sort equations [5].

Given A and B in $\mathsf{Type}(\Gamma)$ we write $A \times B$ for $\Sigma(A, B\mathsf{p})$.

- **Universes.** We have $\mathtt{U}_n$ in $\mathsf{Type}_{n+1}(\Gamma)$ such that $\mathsf{Type}_n(\Gamma) = \mathsf{Elem}(\Gamma, \mathtt{U}_n)$, and the action of substitutions on $\mathsf{Elem}(\Gamma, \mathtt{U}_n)$ is compatible with that on $\mathsf{Type}_n(\Gamma)$.
- **Natural numbers.** We have $\mathtt{N}$ in $\mathsf{Type}_0(\Gamma)$ with *zero* $\mathtt{0}$ in $\mathsf{Elem}(\Gamma, \mathtt{N})$ and *successor* $\mathtt{S}(n)$ in $\mathsf{Elem}(\Gamma, \mathtt{N})$ for n in $\mathsf{Elem}(\Gamma, \mathtt{N})$. Given P in $\mathsf{Type}(\Gamma.\mathtt{N})$, z in $\mathsf{Elem}(\Gamma, P[\mathtt{0}])$, s in $\mathsf{Elem}(\Gamma.\mathtt{N}.P, P(\mathsf{p}, S(\mathsf{q}))\mathsf{p})$, and $n : \mathsf{Elem}(\Gamma, \mathtt{N})$, we have the *elimination* $\mathtt{natrec}(P, z, s, n)$ in $\mathsf{Elem}(\Gamma, P[n])$ with $\mathtt{natrec}(P, z, s, \mathtt{0}) = z$, $\mathtt{natrec}(P, z, s, \mathtt{S}(n)) = s[n, \mathtt{natrec}(P, z, s, n)]$.

A *structured cwf* is a cwf with type formers as above.

A (strict) morphism $\mathcal{M} \to \mathcal{N}$ of cwfs is defined in the evident manner and consists of a functor $F\colon \mathsf{Con}_{\mathcal{M}} \to \mathsf{Con}_{\mathcal{N}}$ and natural transformations $u\colon \mathsf{Type}_{\mathcal{M}} \to \mathsf{Type}_{\mathcal{N}}F$ and $v\colon \mathsf{Elem}_{\mathcal{M}} \to \mathsf{Elem}_{\mathcal{N}}(F, u)$ such that v restricts to types of level n and the terminal context and context extension is preserved strictly. A morphism $\mathcal{M} \to \mathcal{N}$ of structured cwfs additionally preserves the operations of the above type formers. We obtain a category of structured cwfs.

1.2 Internal language of presheaves

For the rest of the paper, we fix a category $\mathcal{C}$ in the lowest Grothendieck universe. As in [2, 23, 21], we will use the language of extensional type theory (with subtypes) to describe constructions in the presheaf topos over $\mathcal{C}$.

In the interpretation of this language, a context is a presheaf A over $\mathcal{C}$, a type B over A is a presheaf over the category of elements of A, and an element of B is a section. A *global type* is a type in the global context, i.e. a presheaf over $\mathcal{C}$. Similarly, a *global element* of a global type is a section of that presheaf.

Given a dependent type B over a type A, we think of B as a family of types $B\,a$ indexed by elements a of A. We have the usual dependent sum $\Sigma(a : A).B(a)$ and dependent product $\Pi(a : A).B\,a$, with projections of $s : \Sigma(a : A).B\,a$ written $s.1$ and $s.2$ and application of $f : \Pi(a : A).B\,a$ to $a : A$ written $f\,a$. We also the categorical pairing $\langle f, g\rangle : X \to \Sigma(a : A).B$ given $f : X \to A$ and $g : \Pi(x : X).B\,(f\,a)$ and other commonly used notations. The hierarchy of Grothendieck universes in the ambient set theory gives rise to a cumulative hierarchy $\mathsf{U}_0, \mathsf{U}_1, \ldots, \mathsf{U}_\omega$ of universes à la Russell. We model propositions as subtypes of a fixed type 1 with unique element tt. We have subuniverses $\Omega_i \subseteq \mathsf{U}_i$ of propositions for $i \in \{0, 1 \ldots, \omega\}$.

When working in this internal language, we refer to the types as "sets" to avoid ambiguity with the types of (internal) cwfs we will be considering.

1.3 Cubical categories with families

We now work internally to presheaves over $\mathcal{C}$. We assume the following:

- an *interval* $\mathbb{I} : \mathsf{U}_0$ with *endpoints* $0, 1 : \mathbb{I}$,
- an object $\mathbb{F} : \mathsf{U}_0$ of *cofibrant propositions* with a monomorphism $[-] : \mathbb{F} \to \Omega_0$.

As in [7, 23], a *partial element* of a set T is given by an element φ in $\mathbb{F}$ and a function $[\varphi] \to T$. We say that a total element v of T extends such a partial element φ, u if we have $[\varphi] \to u\,\mathsf{tt} = v$.

Given $A : \mathbb{I} \to \mathsf{U}_\omega$, we write $\mathsf{hasFill}(A)$ for the set of operations taking as inputs φ in $\mathbb{F}$, $b \in \{0, 1\}$, and a partial section u in $\Pi(i : \mathbb{I}).[\varphi] \vee (i = b) \to A\,i$ and producing an extension of u to a total section in $\Pi(i : \mathbb{I})\,A\,i$. Given a set X and $Y : X \to \mathsf{U}_\omega$, we write $\mathsf{Fill}(X, Y)$ for the set of *filling structures* on Y, producing an element of $\mathsf{hasFill}(Y \circ x)$ for x in $\mathbb{I} \to X$. Given s in $\mathsf{Fill}(X, Y)$ and x, φ, b, u as above, we write $s(x, \varphi, b\ u)$ for the resulting total section in $\Pi(i : \mathbb{I}).Y\,(x\,i)$.

We now interpret the definitions of Section 1.1 in the internal language of the presheaf topos. A *cubical cwf* is a structured cwf denoted as before that additionally has the following *cubical operations and type formers.* Again, all specified operations are required to be stable under substitution.

- **Filling operation.** We have fill in $\mathsf{Fill}(\mathsf{Type}(\Gamma), \lambda A. \mathsf{Elem}(\Gamma, A))$ for Γ in Con. Let us spell out stability under substitution: given $A \colon \mathbb{I} \to \mathsf{Type}(\Gamma)$, φ in $\mathbb{F}$, $b \in \{0,1\}$, u in $\Pi(i : \mathbb{I}). [\varphi] \vee (i = b) \to \mathsf{Elem}(\Gamma, A\,i)$, and σ in $\mathsf{Hom}(\Delta, \Gamma)$ and $r : \mathbb{I}$, we have $(\mathtt{fill}(A, \varphi, b, u)\, r)\sigma = \mathtt{fill}(\lambda i. (A\,i)\sigma, \varphi, b, \lambda i\,x. (u\,i\,x)\sigma)\, r$.

Note that we do not include computation rules for fill at type formers. This corresponds to our decision to treat fill as a non-canonical operation.

- **Dependent path types.** Given A in $\mathbb{I} \to \mathsf{Type}(\Gamma)$ with a_b in $\mathsf{Elem}(\Gamma, Ab)$ for $b \in \{0,1\}$, we have a type $\mathtt{Path}(A, a_0, a_1)$ in $\mathsf{Type}(\Gamma)$, of level n if A is. Given u in $\Pi(i : \mathbb{I}).\mathsf{Elem}(\Gamma, Ai)$, we have the *path abstraction* $\langle\rangle(u)$ in $\mathsf{Elem}(\Gamma, \mathtt{Path}(A, u\,0, u\,1))$. Given p in $\mathsf{Elem}(\Gamma, \mathtt{Path}(A, a_0, a_1))$ and i in $\mathbb{I}$, we have the *path application* $\mathtt{ap}(p, r)$ in $\mathsf{Elem}(\Gamma, Ai)$. These operations satisfy the laws $\mathtt{ap}(p, b) = a_b$ for $b \in \{0,1\}$, $\mathtt{ap}(\langle\rangle(u), i) = u\,i$, and $\langle\rangle(\lambda i. \mathtt{ap}(p, i)) = p$. Thus, every element of $\mathsf{Elem}(\Gamma, \mathtt{Path}(A, a_0, a_1))$ is uniquely of the form $\langle\rangle(u)$ with u in $\Pi(i : \mathbb{I}).\mathsf{Elem}(\Gamma, Ai)$ such that $u\,0 = a_0$ and $u\,1 = a_1$.

Using path types, we define $\mathtt{isContr}(A)$ in $\mathsf{Type}(\Gamma)$ for A in $\mathsf{Type}(\Gamma)$ as well as isEquiv in $\mathsf{Type}(\Gamma.A \to B)$ and Equiv in $\mathsf{Type}(\Gamma)$ for A, B in $\mathsf{Type}(\Gamma)$ as in [6]. These notions are used in the following type former, which extends any partially defined equivalence (given total codomain) to a totally defined function.

- **Glue types.** Given A in $\mathsf{Type}(\Gamma)$, φ in $\mathbb{F}$, T in $[\varphi] \to \mathsf{Type}(\Gamma)$ and $e : [\varphi] \to \mathsf{Elem}(\Gamma, \mathsf{Equiv}(T\mathsf{tt}, A))$, we have the *glueing* $\mathtt{Glue}(A, \varphi, T, e)$ in $\mathsf{Type}(\Gamma)$, equal to T on $[\varphi]$ and of level n if A and T are. We have unglue in $\mathsf{Elem}(\Gamma, \mathsf{Glue}(A, \varphi, T, e) \to A)$ such that $\mathtt{unglue} = \mathtt{fst}(e)\,\mathsf{tt}$ on $[\varphi]$. Given a in $\mathsf{Elem}(\Gamma, A)$ and t in $[\varphi] \to \mathsf{Elem}(\Gamma, T)$ such that $\mathtt{app}(\mathtt{fst}(e)\,\mathsf{tt}, t\,\mathsf{tt}) = a$ on $[\varphi]$, we have $\mathtt{glue}(a, t)$ in $\mathsf{Elem}(\Gamma, \mathsf{Glue}(A, \varphi, T, e))$ equal to t on $[\varphi]$. These operations satisfy $\mathtt{app}(\mathtt{unglue}, \mathtt{glue}(a, t)) = a$ and $\mathtt{glue}(\mathtt{app}(\mathtt{unglue}, u), \lambda x.u) = u$. Thus, every element of $\mathsf{Elem}(\Gamma, \mathtt{Glue}(A, \varphi, T, e))$ is uniquely of the form $\mathtt{glue}(a, t)$ with a and t as above.

The notion of morphism of structured cwfs lifts to an evident notion of morphism of cubical cwfs. We obtain, internally to presheaves over $\mathcal{C}$, a category of cubical cwfs. We now lift this category of cubical cwfs from the internal language to the ambient theory by interpreting it in the global context: externally, a cubical cwf (relative to the chosen base category $\mathcal{C}$, interval $\mathbb{I}$, and cofibrant propositions $\mathbb{F}$) consists of a presheaf Con over $\mathcal{C}$, a presheaf Type over the category of elements of Con, etc.

▶ Remark 1. Fix a cubical cwf as above. Assume that $\mathbb{I}$ has a connection algebra structure and that $\mathbb{F}$ forms a sublattice of Ω_0 that contains the interval endpoint inclusions. As in [6], it is then possible in the above context of the glue type former to construct an element of $\mathsf{Elem}(\Gamma, \mathtt{isEquiv}[\mathtt{unglue}])$. From this, one derives an element of $\mathsf{Elem}(\Gamma, \mathtt{iUnivalence}_n)$ where $\mathtt{iUnivalence}_n = \Pi(\mathtt{U}_n, \mathtt{isContr}(\Sigma(\mathtt{U}_n, \mathtt{Equiv}(\mathtt{q}, \mathtt{qp}))))$ for $n \geq 0$, i.e. univalence is provable. One may also show that the path type applied to constant families $\mathbb{I} \to \mathsf{Type}(\Gamma)$ interprets the rules of identity types of Martin-Löf with the computation rule for the eliminator J replaced by a propositional equality. Thus, we obtain an interpretation of univalent type theory with identity types with propositional computation in any cubical cwf.

2 Two examples of cubical cwfs

In this section we give two examples of cubical cwfs: a term model and a particular cubical cwfs formulated in a constructive metatheory, the latter with extra assumptions on $\mathbb{I}$ and $\mathbb{F}$.

2.1 Term model

We sketch how to give a cubical cwf $\mathcal{T}$ built from syntax, and refer the reader to Appendix A for more details. All our judgments will be indexed by an object X of $\mathcal{C}$ and given a judgment $\Gamma \vdash_X \mathcal{J}$ and $f\colon Y \to X$ in $\mathcal{C}$ we get $\Gamma f \vdash_Y \mathcal{J} f$. Here, f acts on expressions as an implicit substitution, while for substitutions on object variables we will use explicit substitutions.

The forms of judgment are:

$$\Gamma \vdash_X \qquad \Gamma \vdash_X A \qquad \Gamma \vdash_X A = B \qquad \Gamma \vdash_X t : A \qquad \Gamma \vdash_X t = u : A \qquad \sigma\colon \Delta \to_X \Gamma$$

The main rules are given in the appendix. This then induces a cubical cwf $\mathcal{T}$ by taking, say, the presheaf of contexts at stage X to be equivalence classes of Γ for $\Gamma \vdash_X$ where the equivalence relation is judgmental equality.

Some rules are a priori infinitary, but in some cases (such as the one considered in [6]) it is possible to present the rules in a finitary way.

This formal system expresses the laws of cubical cwfs in rule form. It defines the *term model.* Following [29, 24] developed in an intuitionistic framework, we conjecture that this can be interpreted in an arbitrary cubical cwf in the usual way:

▶ **Conjecture 2.** *With chosen parameters $\mathcal{C}, \mathbb{I}, \mathbb{F}$, the cubical cwf $\mathcal{T}$ is initial in the category of cubical cwfs.*

However, our canonicity result is orthogonal to this conjecture: It is a result about the initial model, without need for an explicit description of this model as a term model.

2.2 Developments in presheaves over $\mathcal{C}$

We now assume that $\mathbb{I}$ and $\mathbb{F}$ satisfy the axioms $\mathtt{ax}_1, \dots, \mathtt{ax}_9$ of [23], internally to presheaves over $\mathcal{C}$. We also make an external assumption, namely that the endofunctor on presheaves over $\mathcal{C}$ of exponentiation with $\mathbb{I}$ has a right adjoint R that preserves global types of level n. This is e.g. the case if $\mathbb{I}$ is representable and $\mathcal{C}$ is closed under finite products.

Most of the arguments will be done in the internal language of the presheaf topos. At certain points however, we need to consider the set of global sections of a global type F; we denote this by $\Box F$. We stress that statements involving $\Box$ are external, not to be interpreted in the internal language. Crucially, the adjunction $(-)^{\mathbb{I}} \dashv R$ cannot be made internal [21].

Recall the global type $\mathsf{hasFill} : \mathsf{U}_\omega^{\mathbb{I}} \to \mathsf{U}_\omega$ from Section 1.3. Taking the slice over U_ω, the adjunction $(-)^{\mathbb{I}} \dashv R$ descends to an adjunction between categories of types over U_ω and $\mathsf{U}_\omega^{\mathbb{I}}$. Applying the right adjoint of this adjunction to $\mathsf{hasFill}$, we obtain global $\mathsf{C} : \mathsf{U}_\omega \to \mathsf{U}_\omega$ such that naturally in a global type X with global $Y : X \to \mathsf{U}_\omega$, global elements of $\Pi(x : X^{\mathbb{I}}).\mathsf{hasFill}(Y \circ x)$ are in bijection with global elements of $\Pi(x : X).\mathsf{C}(Y\,x)$. Given a global type X and global $Y : X \to \mathsf{U}_\omega$, we thus have a logical equivalence (maps back and forth)

$$\Box\mathsf{Fill}(X, Y) \longleftrightarrow \Box\Pi(x : X)\,\mathsf{C}(Y\,x) \tag{1}$$

natural in X.[3] Note that C descends to $\mathsf{C} : \mathsf{U}_n \to \mathsf{U}_n$ for $n \geq 0$. We write $\mathsf{U}_i^{\mathsf{fib}} = \Sigma(A : \mathsf{U}_i)\,\mathsf{C}(A)$ for $i \in \{0, 1, \ldots, \omega\}$; we call $\mathsf{U}_i^{\mathsf{fib}}$ a universe of *fibrant sets*. Now set $X = \mathsf{U}_\omega^{\mathsf{fib}}$ and $Y(A, c) = A$ in (1). We trivially have $\Box\Pi(x : X)\,\mathsf{C}(Y\,x)$, thus get

$$\mathsf{fill} : \mathsf{Fill}(\mathsf{U}_\omega^{\mathsf{fib}}, \lambda(A, c).A). \tag{2}$$

This is essentially the counit of the adjunction defining C. Note that [21] use modal extensions of type theory to perform this reasoning internal to presheaves over $\mathcal{C}$.

▶ Remark 3. Internally, a map $\mathsf{Fill}(X, Y) \to \Pi(x : X)\,\mathsf{C}(Y\,x)$ does not generally exist for a set X and $Y : X \to \mathsf{U}_\omega$ as for $X = 1$ one would derive a filling structure for any "homogeneously fibrant" set, which is impossible (see [23, Remark 5.9]). However, from (2) we get a map $\Pi(x : X)\,\mathsf{C}(Y\,x) \to \mathsf{Fill}(X, Y)$ natural in X using closure of filling structures under substitution (see below).

We recall some constructions of [6, 23] in the internal language.

- Given $A : \mathbb{I} \to \mathsf{U}_\omega$ and $a_b : A_b$ for $b \in \{0, 1\}$, *dependent paths* $\mathsf{Path}_A\,a_0\,a_1$ are the set of maps $p : \Pi(i : \mathbb{I}).A\,i$ such that $p\,0 = a_0$ and $p\,1 = a_1$. We use the same notation for non-dependent paths.
- For $A : \mathsf{U}_\omega$, we have a set $\mathsf{isContr}(A)$ of witnesses of *contractibility*, defined using paths.
- Given $A, B : \mathsf{U}_\omega$ with $f : A \to B$, we have the set $\mathsf{isEquiv}(f)$ with elements witnessing that f is an *equivalence*, defined using contractibility of homotopy fibers. We write $\mathsf{Equiv}(A, B) = \Sigma(f : A \to B).\mathsf{isEquiv}(f)$.
- Given $A : \mathsf{U}_\omega$, $\varphi : \mathbb{F}$, $B : [\varphi] \to \mathsf{U}_\omega$, and $e : [\varphi] \to \mathsf{Equiv}(B\,\mathsf{tt}, A)$, the *glueing* $\mathsf{Glue}\,A\,[\varphi \mapsto (B, e)]$ consists of elements $\mathsf{glue}\,a\,[\varphi \mapsto b]$ with $a : A$ and $b : [\varphi] \to B$ such that $e.1\,(b\,\mathsf{tt}) = a$ on $[\varphi]$ and is defined in such a way that $\mathsf{Glue}\,A\,[\varphi \mapsto (B, e)] = T\,\mathsf{tt}$ and $\mathsf{glue}\,a\,[\varphi \mapsto b] = b\,\mathsf{tt}$ on $[\varphi]$. The canonical map $\mathsf{unglue} : \mathsf{Glue}\,A\,[\varphi \mapsto (B, e)] \to A$ is an equivalence.

These operations are valued in U_n if their inputs are. We further recall basic facts from [23] about filling structures in the internal language.

- Filling structures are closed under substitution: given $f : X' \to X$ and $Y : X \to \mathsf{U}_\omega$, any element of $\mathsf{Fill}(X, Y)$ induces an element of $\mathsf{Fill}(X', Y \circ f)$, naturally in X'.
- Filling structures are closed under exponentiation: given sets S, X and $Y : X \to \mathsf{U}_\omega$, any element of $\mathsf{Fill}(X, Y)$ induces an element of $\mathsf{Fill}(X^S, \lambda x.\Pi(s : S).Y(x\,s))$, naturally in S.
- Filling structures are closed under $\Pi, \Sigma, \mathsf{Path}, \mathsf{Glue}$. E.g. for dependent products, given $A : \Gamma \to \mathsf{U}_\omega$ with $\mathsf{Fill}(\Gamma, A)$ and $B : \Pi(\rho : \Gamma).A\,\rho \to \mathsf{U}_\omega$ with $\mathsf{Fill}(\Sigma(\rho : \Gamma).A\,\rho, \lambda(\rho, a).B\,\rho\,a)$, we have $\mathsf{Fill}(\Gamma, \lambda(\rho : \Gamma).\Pi(a : A\,\rho).B\,\rho\,a)$.

From the last point, we deduce using that C is closed under $\Pi, \Sigma, \mathsf{Path}, \mathsf{Glue}$, and that $\mathsf{C}(A)$ implies $\mathsf{C}(A^S)$ for $A, S : \mathsf{U}_\omega$.[4] Let us explain this in the case of dependent products. We apply (1) with a suitable "generic context" $X = \Sigma((A, c) : \mathsf{U}_\omega^{\mathsf{fib}}).A \to \mathsf{U}_\omega^{\mathsf{fib}}$ and $Y((A, c), \langle B, d\rangle) = \Pi(a : A).B\,a$. Using the map of Remark 3 and closure of filling structures under dependent product, we have $\Box\mathsf{Fill}(X, Y)$ and can conclude $\mathsf{C}(\Pi(a : A).B\,a)$ for $(A, c) : \mathsf{U}_\omega^{\mathsf{fib}}$ and $\langle B, d\rangle : A \to \mathsf{U}_\omega^{\mathsf{fib}}$. Note that in the case of Glue with $(A, c) : \mathsf{U}_\omega^{\mathsf{fib}}$, $\varphi : \mathbb{F}$, $\langle B, d\rangle : [\varphi] \to \mathsf{U}_\omega^{\mathsf{fib}}$, and $e : [\varphi] \to \mathsf{Equiv}(B\,\mathsf{tt}, A)$, naturality of the forward map of (1) is needed to see that the element $c : \mathsf{C}(\mathsf{Glue}\,A\,[\varphi \mapsto (B, e)])$ constructed in the same fashion as above for dependent products equals $d\,\mathsf{tt} : C(B\,\mathsf{tt})$ on $[\varphi]$.

---

[3] We record only the logical equivalence instead of an isomorphism so that it will be easier to apply our constructions in situations where the right adjoint R fails to exist, see Appendix D. Naturality is only used at a one point below, for the forward map, to construct suitable elements of C applied to glueings.

[4] Note that naturality in S of the latter operation is used in substitutional stability of universes in the sconing in Section 3.

As in [6, 23, 21], glueing shows $\mathsf{Fill}(1, \mathsf{U}^{\mathsf{fib}}_n)$ for $n \geq 0$. Using (1), we conclude $\mathsf{C}(\mathsf{U}^{\mathsf{fib}}_n)$.

Let $\mathbb{N}$ denote the natural number object in presheaves over $\mathcal{C}$, the constant presheaf with value the natural numbers. From [6, 23], we have $\mathsf{Fill}(1, \mathbb{N})$. Using (1), we conclude $\mathsf{C}(\mathbb{N})$.

We justify fibrant indexed inductive sets in Appendix B.

2.3 Standard model

Making the same assumptions on $\mathcal{C}, \mathbb{I}, \mathbb{F}$ as in Section 2.2, we can now specify the standard model $\mathcal{S}$ of cubical type theory in the sense of the current paper as a cubical cwf (with respect to parameters $\mathcal{C}, \mathbb{I}, \mathbb{F}$) purely using the internal language of the presheaf topos. The cwf is induced by the family over $\mathsf{U}^{\mathsf{fib}}_\omega$ given by the first projection as follows.

- The category of contexts is U_ω, with $\mathsf{Hom}(\Delta, \Gamma)$ the functions from Δ to Γ.
- The types over Γ are maps from Γ to $\mathsf{U}^{\mathsf{fib}}_\omega$; a type $\langle A, p\rangle$ is of level n if A is in $\Gamma \to \mathsf{U}_n$. This is clearly functorial in Γ.
- The elements of $\langle A, p\rangle : \Gamma \to \mathsf{U}^{\mathsf{fib}}_\omega$ are $\Pi(\rho : \Gamma). A\,\rho$. This is clearly functorial in Γ.
- The terminal context is given by 1.
- The context extension of Γ by $\langle A, p\rangle$ is given by $\Sigma(\rho : \Gamma). A\,\rho$, with p, q given by projections and substitution extension given by pairing.

We briefly go through the necessary type formers and operations, omitting evident details.

- The dependent product of $\langle A, c\rangle : \Gamma \to \mathsf{U}^{\mathsf{fib}}_\omega$ and $\langle B, d\rangle : \Sigma(\rho : \Gamma). A\,\rho \to \mathsf{U}^{\mathsf{fib}}_\omega$ is $\langle \lambda\rho.\Pi(a : A\,\rho). B(\rho, a), e\rangle$ where $e\,\rho : \mathsf{C}(\Pi(a : A\,\rho). B(\rho, a))$ is induced by $c\,\rho : \mathsf{C}(A\,\rho)$ and $d\,\rho\,a : \mathsf{C}(B(\rho, a))$ for $a : A$ as discussed above.
- The dependent sum of $\langle A, c\rangle : \Gamma \to \mathsf{U}^{\mathsf{fib}}_\omega$ and $\langle B, d\rangle : \Sigma(\rho : \Gamma). A\,\rho \to \mathsf{U}^{\mathsf{fib}}_\omega$ is $\langle \lambda\rho.\Sigma(a : A\,\rho). B(\rho, a), e\rangle$ where e is induced by c and d.
- The universe $\mathtt{U}_n : \Gamma \to \mathsf{U}^{\mathsf{fib}}_{n+1}$ is constantly $(\mathsf{U}^{\mathsf{fib}}_n, c)$ with $c : \mathsf{C}(\mathsf{U}^{\mathsf{fib}}_n)$ as above.
- The natural number type $\mathtt{N} : \Gamma \to \mathsf{U}^{\mathsf{fib}}_0$ is constantly $(\mathbb{N}, c)$ with $c : \mathsf{C}(\mathbb{N})$ as above. The zero and successor constructors and eliminator are given by the corresponding features of the natural number object $\mathbb{N}$.

We now turn to the cubical aspects.

- The filling operation $\mathtt{fill} : \mathsf{Fill}(\Gamma \to \mathsf{U}^{\mathsf{fib}}_\omega, \lambda\langle A, p\rangle.\Pi(\rho : \Gamma). A\,\rho)$ is derived from (2) by closure of filling structures under exponentiation.
- Given $\langle A, c\rangle : \mathbb{I} \to \Gamma \to \Sigma(A : \mathsf{U}_\omega)\,\mathsf{C}(A)$ and $a_b : \Pi(\rho : \Gamma). A\,b\,\rho$ for $b \in \{0, 1\}$, we define $\mathtt{Path}(A, a_0, a_1) : \Gamma \to \Sigma(A : \mathsf{U}_\omega)\,\mathsf{C}(A)$ as $\langle \Pi(\rho : \Gamma).\mathsf{Path}_{\lambda i. A\,i\,\rho}\,c_0\,c_1), d\rangle$ where $d\,\rho : \mathsf{C}(\mathsf{Path}_{\lambda i. A\,i\,\rho}\,c_0\,c_1))$ induced by $\lambda i. c\,i\,\rho$.

Before defining glue types, we note that the notions $\mathtt{isContr}$ and $\mathtt{isEquiv}$ in the cubical cwf we are defining correspond to the notions $\mathsf{isContr}$ and $\mathsf{isEquiv}$. For example, given a type $A : \Gamma \to \mathsf{U}^{\mathsf{fib}}_\omega$, then the elements of $\mathtt{isContr}(A)$, given by $\Pi(\rho : \Gamma).\,\mathtt{isContr}(A).1\,\rho$, are in bijection with $\Pi(\rho : \Gamma).\mathsf{isContr}(A.1\,\rho)$ naturally in Γ.

- Given $\langle A, c\rangle : \Gamma \to \mathsf{U}^{\mathsf{fib}}_\omega$, $\varphi : \mathbb{F}$, $\langle T, d\rangle : [\varphi] \to \Gamma \to \mathsf{U}^{\mathsf{fib}}_\omega$ and $e : [\varphi] \to \mathtt{Equiv}(T\,\mathsf{tt}, A)$, we define $\mathtt{Glue}(\langle A, c\rangle, \varphi, \langle T, d\rangle, e) : \Gamma \to \mathsf{U}^{\mathsf{fib}}_\omega$ as $\lambda\rho.(\mathsf{Glue}\,(A\,\rho)[\varphi \mapsto (T\,\mathsf{tt}\,\rho, e'\,\mathsf{tt}\,\rho)], q\,\rho)$ where $e'\,\mathsf{tt}\,\rho : \mathsf{Equiv}(T\,\mathsf{tt}\,\rho, A\,\rho)$ is induced by $e\,\mathsf{tt}\,\rho$ and $q\,\rho$ is induced by $c\,\rho$ and $\lambda x. d\,x\,\rho$.

We have thus verified the following statement.

► **Theorem 4.** *Assuming the parameters $\mathcal{C}, \mathbb{I}, \mathbb{F}$ satisfy the assumptions of Section 2.2, the standard model $\mathcal{S}$ forms a cubical cwf.*

3 Sconing

We make the same assumptions on our parameters $\mathcal{C}, \mathbb{I}, \mathbb{F}$ as in Section 2.2. Let $\mathcal{M}$ be a cubical cwf (with respect to these parameters) denoted $\mathsf{Con}, \mathsf{Hom}, \ldots$ as in Section 1.3. We assume that $\mathcal{M}$ is *size-compatible* with the standard model, by which we mean $\mathsf{Hom}(\Delta, \Gamma) : \mathsf{U}_\omega$ for all Γ, Δ and $\mathsf{Elem}(\Gamma, A) : \mathsf{U}_i$ for $i \in \{0, 1, \ldots, \omega\}$ and all Γ and $A : \mathsf{Type}_i(\Gamma)$. We will then define a new cubical cwf $\mathcal{M}^*$ denoted $\mathsf{Con}^*, \mathsf{Hom}^*, \ldots$, the *(Artin) glueing* of $\mathcal{M}$ with the standard model $\mathcal{S}$ along an (internal) global sections functor, i.e. the *sconing* of $\mathcal{M}$.

Recall from Section 1.3 the operation `fill` of $\mathcal{M}$. Instantiating it to the terminal context, we get $\Box\mathsf{Fill}(\mathsf{Type}(1), \lambda A.\mathsf{Elem}(1, A))$. Using the forward direction of Equation (1), we thus have an internal operation $k : \Pi(A : \mathsf{Type}(1)).\mathsf{C}(\mathsf{Elem}(1, A))$.

From now on, we will work in the internal language of presheaves over $\mathcal{C}$. We start by defining a global sections operation $|{-}|$ mapping contexts, types, and elements of $\mathcal{M}$ to those of $\mathcal{S}$.

- Given $\Gamma : \mathsf{Con}$, we define $|\Gamma| : \mathsf{U}_\omega$ as the set of substitutions $\mathsf{Hom}(1, \Gamma)$. Given a substitution $\sigma : \mathsf{Hom}(\Delta, \Gamma)$, we define $|\sigma| : |\Delta| \to |\Gamma|$ as $|\sigma|\rho = \sigma\rho$. This evidently defines a functor.
- Given $A : \mathsf{Type}(\Gamma)$, we define $|A| : |\Gamma| \to \mathsf{U}^{\mathsf{fib}}_\omega$ as $|A|\,\rho = (\mathsf{Elem}(1, A\rho), k\,(A\,\rho))$. This evidently natural in Γ. If A is of level n, then $|A| : |\Gamma| \to \mathsf{U}^{\mathsf{fib}}_n$.
- Given $a : \mathsf{Elem}(\Gamma, A)$ we define $|a| : \Pi(\rho : \Gamma).\,(|A|\,\rho).1$ as $|a|\,\rho = a\rho$. This is evidently natural in Γ.

Note that $|{-}|$ preserves the terminal context and context extension up to canonical isomorphism in the category of contexts. One could thus call $|{-}|$ an (internal) *pseudomorphism* cwfs from $\mathcal{M}$ to $\mathcal{S}$. The sconing $\mathcal{M}^*$ will be defined as essentially the (Artin) glueing along this pseudomorphism, but we will be as explicit as possible and not define (Artin) glueing at the level of generality of an abstract pseudomorphism.

For convenience, we also just write $|A| : |\Gamma| \to \mathsf{U}_\omega$ instead of $\lambda\rho.\,(|A|\,\rho).1$, implicitly applying the first projection. We also write just $|A|$ for $|A|\ |()|$ if Γ is the terminal context.

3.1 Contexts, substitutions, types and elements

We start by defining the cwf $\mathcal{M}^*$.

- A context $(\Gamma, \Gamma') : \mathsf{Con}^*$ consists of a context $\Gamma : \mathsf{Con}$ in $\mathcal{M}$ and a family Γ' over $|\Gamma|$ (which in the context of Artin glueing should be thought of as a substitution in $\mathcal{S}$ from some context to $|\Gamma|$). We think of Γ' as a *proof-relevant computability predicate*. A substitution $(\sigma, \sigma') : \mathsf{Hom}^*((\Delta, \Delta'), (\Gamma, \Gamma'))$ consists of a substitution $\sigma : \Delta \to \Gamma$ in $\mathcal{M}$ and a map $\sigma' : \Pi(\nu : |\Delta|).\Delta'(\nu) \to \Gamma'(\sigma\nu)$. This evidently has the structure of a category.
- A type $(A, A') : \mathsf{Type}^*(\Gamma, \Gamma')$ consists of a type $A : \mathsf{Type}(\Gamma)$ in $\mathcal{M}$ and

  $$A' : \Pi(\rho : |\Gamma|)(\rho' : \Gamma'\,\rho).|A|\,\rho \to \mathsf{U}^{\mathsf{fib}}_\omega.$$

  We think of A' as a fibrant *proof-relevant computability family* on A. In the abstract context of Artin glueing for cwfs, we should think of it as an element of $\mathsf{Type}(\Sigma(\rho : |\Gamma|)(\rho' : \Gamma'\,\rho).|A|\,\rho)$ in $\mathcal{S}$, but this point of view is not compatible with the universes à la Russell we are going to model. Recalling $\mathsf{U}^{\mathsf{fib}}_\omega = \Sigma(X : \mathsf{U}_\omega).C(X)$, we also write $\langle A', \mathsf{fib}_{A'}\rangle$ instead of A' if we want to directly access the family and split off its proof of fibrancy.
  The type (A, A') is of level n if A and A' are.
  The action of a substitution $(\sigma, \sigma') : \mathsf{Hom}^*((\Delta, \Delta'), (\Gamma, \Gamma'))$ on (A, A') is given by

  $$(A\sigma, \lambda\nu\,\nu'\,a.A'\,(\sigma\nu)\,(\sigma'\,\nu\,\nu')\,a).$$

- An element $(a, a') : \mathsf{Elem}^*((\Gamma, \Gamma'), (A, \langle A', \mathsf{fib}_{A'}\rangle))$ consists of $a : \mathsf{Elem}(\Gamma, A)$ in $\mathcal{M}$ and

  $$a' : \Pi(\rho : |\Gamma|)(\rho' : \Gamma'\,\rho).\, A'(\rho, \rho', a\rho).$$

  In the context of Artin glueing (with types in $\mathcal{M}^*$ presented correspondingly), this should be thought of as an element $a' : \mathsf{Elem}(\Sigma(\rho : |\Gamma|).\Gamma'\,\rho, \lambda(\rho, \rho').A'(\rho, \rho', |a|\,\rho))$ of $\mathcal{S}$.
  The action of a substitution $(\sigma, \sigma') : \mathsf{Hom}^*((\Delta, \Delta'), (\Gamma, \Gamma'))$ on the element (a, a') is given by $(a\sigma, \lambda\nu\,\nu'.a'\,\sigma\nu\,(\sigma'\,\nu\,\nu'))$.
- The terminal context is given by $(1, 1')$ defined by $1'\,() = 1$.
- The extension in $\mathcal{M}^*$ of a context (Γ, Γ') by a type (A, A') is given by $(\Gamma.A, (\Gamma.A)')$ where $(\Gamma.A')(\rho, a) = \Sigma(\rho' : \Gamma'\,\rho).\,(A'\,\rho\,\rho'\,a).1$. The projection $\mathsf{p}^* : \mathsf{Hom}^*((\Gamma, \Gamma').(A, A'), (\Gamma, \Gamma'))$ is $(\mathsf{p}, \mathsf{p}')$ where $\mathsf{p}'\,(\rho, a)\,(\rho', a') = \rho'$ and the generic term $\mathsf{q}^* : \mathsf{Elem}((\Gamma, \Gamma').(A, A')\mathsf{p}^*)$ is $(\mathsf{q}, \mathsf{q}')$ where $\mathsf{q}'\,(\rho, a)\,(\rho', a') = a'$. The extension of $(\sigma, \sigma') : \mathsf{Hom}^*((\Delta, \Delta'), (\Gamma, \Gamma'))$ with $(a, a') : \mathsf{Elem}^*((\Delta, \Delta'), (A, A')(\sigma, \sigma'))$ is $((\sigma, a), \lambda\nu\,\nu'.(\sigma'\,\nu\,\nu', a'\,\nu\,\nu'))$.

3.2 Type formers and operations

3.2.1 Dependent products

Let $(A, \langle A', \mathsf{fib}_{A'}\rangle) : \mathsf{Type}^*(\Gamma, \Gamma')$ and $(B, \langle B', \mathsf{fib}_{B'}\rangle) : \mathsf{Type}^*((\Gamma, \Gamma').(A, \langle A', \mathsf{fib}_{A'}\rangle))$. We define the dependent product $\Pi^*((A, \langle A', \mathsf{fib}_{A'}\rangle), (B, \langle B', \mathsf{fib}_{B'}\rangle)) = (\Pi(A, B), \langle \Pi(A, B)', \mathsf{fib}_{\Pi(A,B)'}\rangle)$ where

$$\Pi(A, B)'(\rho, \rho', f) = \Pi(a : |A|\,\rho)(a' : A'\,\rho\,\rho'\,a).B'\,(\rho, a)\,(\rho', a')\,(\mathsf{app}(f, a))$$

and $\mathsf{fib}_{\Pi(A,B)'}(\rho, \rho', f)$ is given by closure of C under dependent product applied to $(|A|\,\rho).2$, $\mathsf{fib}_{A'}\,\rho\,\rho'\,a$ for $a : |A|\,\rho$, and $\mathsf{fib}_{B'}\,(\rho, a)\,(\rho', a')\,(\mathsf{app}(f, a))$ for additionally $a' : A'\,\rho\,\rho'\,a$.

Given an element (b, b') of $(B, \langle B', d\rangle))$ in $\mathcal{M}^*$, we define the abstraction $\mathtt{lam}^*(b, b') = (\mathtt{lam}(b), \mathtt{lam}(b)')$ where $\mathtt{lam}(b)'\,\rho\,\rho'\,a\,a' = b'\,(\rho, a)\,(\rho', a')$.

Given elements (f, f') of $\Pi^*((A, \langle A', c\rangle), (B, \langle B', d\rangle))$ and (a, a') of $(A, \langle A', \mathsf{fib}_{A'}\rangle))$ in $\mathcal{M}^*$, we define the application $\mathtt{app}^*((f, f'), (a, a')) = (\mathtt{app}(f, a), \mathtt{app}(f, a)')$ where $\mathtt{app}(f, a)'\,\rho\,\rho' = f'\,\rho\,\rho'\,a\rho\,(a'\,\rho\,\rho')$.

3.2.2 Dependent sums

Let $(A, \langle A', \mathsf{fib}_{A'}\rangle) : \mathsf{Type}^*(\Gamma, \Gamma')$ and $(B, \langle B', \mathsf{fib}_{B'}\rangle) : \mathsf{Type}^*((\Gamma, \Gamma').(A, \langle A', \mathsf{fib}_{A'}\rangle))$. We define the dependent sum $\Sigma^*((A, \langle A', \mathsf{fib}_{A'}\rangle), (B, \langle B', \mathsf{fib}_{B'}\rangle)) = (\Sigma(A, B), \langle \Sigma(A, B)', \mathsf{fib}_{\Sigma(A,B)'}\rangle)$ where

$$\Sigma(A, B)'\,\rho\,\rho'\,(\mathtt{pair}(a, b)) = \Sigma(a' : A'\,\rho\,\rho'\,a).B'\,(\rho, a)\,(\rho', a')\,b$$

and $\mathsf{fib}_{\Sigma(A,B)'}\,\rho\,\rho'\,(\mathtt{pair}(a, b))$ is given by closure of C under dependent sum applied to $\mathsf{fib}_{A'}\,\rho\,\rho'\,a$ and $\mathsf{fib}_{B'}\,(\rho, a)\,(\rho', a')\,b$.

Given elements (a, a') of $(A, \langle A', \mathsf{fib}_{A'}\rangle)$ and (b, b') of $(B, \langle B', \mathsf{fib}_{B'}\rangle)[(a, a')]$ in $\mathcal{M}^*$, we define the pairing $\mathtt{pair}^*((a, a'), (b, b')) = (\mathtt{pair}(a, b), \langle a', b'\rangle)$.

Given an element $(\mathtt{pair}(a, b), \langle a', b'\rangle)$ of $\Sigma^*((A, \langle A', \mathsf{fib}_{A'}\rangle), (B, \langle B', \mathsf{fib}_{B'}\rangle))$ in $\mathcal{M}^*$, we define the projections $\mathtt{fst}^*(\mathtt{pair}(a, b), \langle a', b'\rangle) = (a, a')$ and $\mathtt{snd}^*(\mathtt{pair}(a, b), \langle a', b'\rangle) = (b, b')$.

3.2.3 Universes

We define the universe $\mathtt{U}^*_n : \mathsf{Type}^*(\Gamma,\Gamma')$ as $\mathtt{U}^*_n = (\mathtt{U}_n, \langle \mathtt{U}'_n, \mathsf{fib}_{\mathtt{U}'_n}\rangle)$ where $\mathtt{U}'_n\,\rho\,\rho'\,A = |A|\,\rho \to \mathsf{U}^{\mathsf{fib}}_n$ and $\mathsf{fib}_{\mathtt{U}'_n}\,\rho\,\rho'\,A$ is given by $\mathsf{C}(\mathsf{U}^{\mathsf{fib}}_n)$ and closure of C under exponentiation (note that fibrancy of $|A|\,\rho$ is not used). We have carefully chosen our definitions so that we get $\mathsf{Elem}^*((\Gamma,\Gamma'),\mathtt{U}^*_n) = \mathsf{Type}^*_n(\Gamma,\Gamma')$ and see that this identity is compatible with the action in $\mathcal{M}^*$ of substitution on both sides.

3.2.4 Natural numbers

As per Appendix B, we have a fibrant indexed inductive set $\mathbb{N}' : |\mathtt{N}| \to \mathsf{U}^{\mathsf{fib}}_0$ (where $\mathtt{N} : \mathsf{Type}_0(1)$, hence $|\mathtt{N}| : \mathsf{U}_0$) with constructors $\mathsf{0}' : \mathbb{N}'\,\mathtt{0}$ and $\mathsf{S}' : \Pi(n : |\mathtt{N}|\,\rho).\mathbb{N}'\,n \to \mathbb{N}'\,(\mathtt{S}\,n)$. In context $(\Gamma,\Gamma') : \mathsf{Con}^*$, we then define $\mathtt{N}^* = (\mathtt{N}, \lambda\rho\,\rho'.\mathbb{N}')$. We have $\mathtt{0}^* = (\mathtt{0}, \lambda\rho\,\rho'.\mathsf{0}')$ and $\mathtt{S}^*(n,n') = (\mathtt{S}(n), \lambda\rho\,\rho'.\mathsf{S}'\,n\rho\,n')$ for $(n,n') : \mathsf{Elem}^*((\Gamma,\Gamma'),\mathtt{N}^*)$.

Given $(P,P') : \mathsf{Type}((\Gamma,\Gamma').\mathtt{N}^*)$ with

$$(z,z') : \mathsf{Elem}^*((\Gamma,\Gamma')(P,P')[\mathtt{0}^*]), \quad (s,s') : \mathsf{Elem}^*((\Gamma,\Gamma').\mathtt{N}^*.(P,P'),(P,P')(\mathsf{p},S^*(\mathsf{q}))\mathsf{p})$$

and $(n,n') : \mathsf{Elem}^*((\Gamma,\Gamma'),\mathtt{N}^*)$, we define the elimination

$$\mathtt{natrec}^*((P,P'),(z,z'),(s,s'),(n,n')) = (\mathtt{natrec}(P,z,s,n), \lambda\rho\,\rho'.h'\,n\rho\,(n'\,\rho\,\rho'))$$

where $h' : \Pi(m : |\mathtt{N}|)(m' : \mathbb{N}'\,m).P'\,(\rho,m)\,(\rho',m')\,(\mathtt{natrec}(P\rho^+, z\rho, s\rho^{+++}, m))$ is given by induction on $\mathbb{N}'$ with defining equations

$$h'\,\mathtt{0}\,\mathsf{0}' = z'\,\rho\,\rho', \qquad h'\,(\mathtt{S}(n))\,(\mathsf{S}'\,n\,n') = s'\,(\rho, n, \mathtt{natrec}(P,z,s,n))\,(\rho',n',h'\,n\,n').$$

3.2.5 Dependent paths

Let $\langle A, A'\rangle : \mathbb{I} \to \mathsf{Type}^*(\Gamma,\Gamma')$ and $(a_b,a'_b) : \mathsf{Elem}^*((\Gamma,\Gamma'),(A\,b,A'\,b))$ for $b \in \{0,1\}$. We then define the dependent path type $\mathtt{Path}^*(\langle A,A'\rangle,(a_0,a'_0),(a_1,a'_1)) : \mathsf{Type}^*(\Gamma,\Gamma')$ as the tuple $(\mathtt{Path}(A,a_0,a_1), \langle \mathtt{Path}(A,a_0,a_1)', \mathsf{fib}_{\mathtt{Path}(A,a_0,a_1)'}\rangle)$ where

$$\mathtt{Path}(A,a_0,a_1)'\,\rho\,\rho'\,(\langle\rangle(u)) = \mathsf{Path}_{\lambda(i:\mathbb{I}).\,(A'\,i\,\rho\,\rho'\,(u\,i)).1}(a'_0\,\rho\,\rho')(a'_1\,\rho\,\rho')$$

and $\mathsf{fib}_{\mathtt{Path}(A,a_0,a_1)'}\,\rho\,\rho'\,(\langle\rangle(u))$ is closure of C under Path applied to $(A'\,i\,\rho\,\rho'\,(u\,i)).2$ for $i : \mathbb{I}$.

Given $\langle u,u'\rangle : \Pi(i : \mathbb{I}).\mathsf{Elem}^*((\Gamma,\Gamma'),(A\,i,A'\,i))$, we define the path abstraction as $\langle\rangle^*(\langle u,u'\rangle) = (\langle\rangle(u), \lambda\rho\,\rho'\,i.u'\,i\,\rho\,\rho')$.

Given $(p,p') : \mathsf{Elem}^*((\Gamma,\Gamma'),\mathtt{Path}^*(\langle A,A'\rangle,(a_0,a'_0),(a_1,a'_1)))$ and $i : \mathbb{I}$, we define the path application $\mathtt{ap}^*(p,i) = (\mathtt{ap}(p,i), \lambda\rho\,\rho'.u'\,\rho\,\rho'\,i)$.

3.2.6 Filling operation

Given $\langle A,A'\rangle : \mathbb{I} \to \mathsf{Type}^*(\Gamma,\Gamma')$, $\varphi : \mathbb{F}$, $b \in \{0,1\}$, and $\langle u,u'\rangle : \Pi(i : \mathbb{I}).[\varphi] \vee (i = b) \to \mathsf{Elem}^*((\Gamma,\Gamma'),(A\,i,A'\,i))$, we have to extend u to

$$\mathtt{fill}^*(\langle A,A'\rangle,\varphi,b,\langle u,u'\rangle) : \Pi(i : \mathbb{I}).\mathsf{Elem}^*((\Gamma,\Gamma'),(A\,i,A'\,i)).$$

We define $\mathtt{fill}^*(\langle A,A'\rangle,\varphi,b,\langle u,u'\rangle) = \langle \mathtt{fill}(A,\varphi,b,u), \mathtt{fill}(A,\varphi,b,u)'\rangle$ where

$$\mathtt{fill}(A,\varphi,b,u)'\,i\,\rho\,\rho' : A'\,i\,\rho\,\rho'\,(\mathtt{fill}(A,\varphi,b,u)\,i)\rho$$

is defined using fill from (2) as

$$\mathtt{fill}(A,\varphi,b,u)'\,i\,\rho\,\rho' = \mathsf{fill}(\lambda i.A'\,i\,\rho\,\rho'\,(\mathtt{fill}(A,\varphi,b,u)\,i)\rho, \varphi, b, \lambda i\,x.u'\,i\,x\,\rho\,\rho').$$

3.2.7 Glue types

Before defining the glueing operation in $\mathcal{M}^*$, we will develop several lemmas relating notions such as contractibility and equivalences in $\mathcal{M}$ with the corresponding notions of Section 2.2. Given $f : \mathsf{Elem}(\Gamma, A \to B)$ in $\mathcal{M}$, we write $|f| : \Pi(\rho : |\Gamma|).|A|\,\rho \to |B|\,\rho$ for $|f|\,\rho\,a = \mathtt{app}(f\rho, a)$. This notation overlaps with the action of $|-|$ on elements, but we will not use that one here.

Just in this subsection, we will use the alternative definition via given left and right homotopy inverses instead of contractible homotopy fibers of both equivalences $\mathtt{Equiv}$ in the cubical cwf $\mathcal{M}$ and equivalences Equiv in the (current) internal language. In both settings, there are maps back and forth to the usual definition, which are furthermore natural in the context in the case of the cubical cwf $\mathcal{M}$. The statements we will prove are then also valid for the usual definition.

▶ **Lemma 5.** *Given* $f : \mathsf{Elem}(\Gamma, A \to B)$ *in* $\mathcal{M}$ *with* $\mathsf{Elem}(\Gamma, \mathtt{isEquiv}(f))$*, we have* $\Pi(\rho : |\Gamma|).\mathsf{isEquiv}(|f|\,\rho)$*. This is natural in* Γ*.*

Proof. A (left or right) homotopy inverse $g : \mathsf{Elem}(\Gamma, B \to A)$ to f in $\mathcal{M}$ becomes a (left or right, respectively) homotopy inverse $|g|\,\rho$ to $|f|\,\rho$ for $\rho : |\Gamma|$. ◀

▶ **Lemma 6.** *Given* $(f, f') : \mathsf{Elem}((\Gamma, \Gamma'), (A, A') \to (B, B'))$ *in* $\mathcal{M}^*$*, the following statements are logically equivalent, naturally in* (Γ, Γ')*:*

$$\mathsf{Elem}((\Gamma, \Gamma'), \mathtt{isEquiv}^*(f, f')), \tag{3}$$

$$\mathsf{Elem}(\Gamma, \mathtt{isEquiv}(f)) \times \Pi(\rho : |\Gamma|)(\rho' : \Gamma'\,\rho).\mathsf{isEquiv}(\Sigma_{|f|\,\rho} f'\,\rho\,\rho'), \tag{4}$$

$$\mathsf{Elem}(\Gamma, \mathtt{isEquiv}(f)) \times \Pi(\rho : |\Gamma|)(\rho' : \Gamma'\,\rho)(a : |A|\,\rho).\mathsf{isEquiv}(f'\,\rho\,\rho'\,a) \tag{5}$$

where $\Sigma_{|f|\,\rho} f'\,\rho\,\rho' : \Sigma(a : |A|\,\rho)\,A'\,\rho\,\rho'\,a \to \Sigma(b : |B|\,\rho)\,B'\,\rho\,\rho'\,b$*.*

Proof. Let us only look at homotopy left inverses.

For (3) → (4), a homotopy left inverse (g, g') to (f, f') in $\mathcal{M}^*$ gives a homotopy left inverse $\Sigma_{|g|\,\rho} g'\,\rho\,\rho'$ to $\Sigma_{|f|\,\rho} f'\,\rho\,\rho'$ for all ρ, ρ'.

For (4) → (5), we use Lemma 5 and note that a fiberwise map over an equivalence is a fiberwise equivalence exactly if it is an equivalence on total spaces (the corresponding statement for identity types instead of paths is [33, Theorem 4.7.7]).

For (5) → (3), given a homotopy left inverse g to the equivalence f in $\mathcal{M}$ and a homotopy left inverse $\overline{g}'\,\rho\,\rho'\,a : B'\,\rho\,\rho'\,(|f|\,a) \to A'\,\rho\,\rho'\,a$ to $f'\,\rho\,\rho'\,a$ for all ρ, ρ', a, we use Lemma 5 to transpose $\overline{g}'$ to the second component $g'\,\rho\,\rho'\,b : B'\,\rho\,\rho'\,b \to A'\,\rho\,\rho'\,(|g|\,b)$ for all ρ, ρ', b of a homotopy left inverse (g, g') to (f, f') in $\mathcal{M}^*$. ◀

We can now define glue types in $\mathcal{M}^*$. Let $(A, A') : \mathsf{Type}(\Gamma, \Gamma')$, $\varphi : \mathbb{F}$, $b \in \{0, 1\}$, $\langle T, T' \rangle : [\varphi] \to \mathsf{Type}(\Gamma, \Gamma')$, and $\langle e, e' \rangle : [\varphi] \to \mathsf{Elem}((\Gamma, \Gamma'), \mathtt{Equiv}^*((T\mathsf{tt}, T'\mathsf{tt}), (A, A')))$.

We define $\mathtt{Glue}^*((A, A'), \varphi, b, \langle T, T' \rangle, \langle e, e' \rangle) = (\mathtt{Glue}(A, \varphi, b, T, e), \langle G', \mathsf{fib}_{G'} \rangle)$ where

$$G'\,\rho\,\rho'\,(\mathtt{glue}(a, t)) = \mathsf{Glue}\,(A'\,\rho\,\rho'\,a).1\,[\varphi \mapsto (T'\,\mathsf{tt}\,\rho\,\rho'\,(t\,\mathsf{tt}), ((e'\,\mathsf{tt}\,\rho\,\rho').1\,(t\,\mathsf{tt}), w\,\mathsf{tt}\,\rho\,\rho'))]$$

using the witness $w\,\mathsf{tt}\,\rho\,\rho'$ that $(e'\,\mathsf{tt}\,\rho\,\rho').1\,(t\,\mathsf{tt})$ is an equivalence provided by the direction from (3) to (5) of Lemma 6 and $\mathsf{fib}_{G'}\,\rho\,\rho'\,(\mathtt{glue}(a, t))$ is given by closure of C under Glue applied to $(A'\,\rho\,\rho'\,a).2$ and $T'\,\mathsf{tt}\,\rho\,\rho'\,(t\,\mathsf{tt})$ on $[\varphi]$.

We define $\mathtt{unglue}^* = (\mathtt{unglue}, \mathtt{unglue}')$ where $\mathtt{unglue}'\,\rho\,\rho'\,(\mathtt{glue}(a, t)) = \mathsf{unglue}$.

Given $(a, a') : \mathsf{Elem}((\Gamma.\Gamma'), (A, A'))$ and $(t, t') : [\varphi] \to \mathsf{Elem}((\Gamma, \Gamma'), (T\,\mathsf{tt}, T'\,\mathsf{tt}))$ such that $\mathtt{app}^*(\mathtt{fst}^*(e, e')\,\mathsf{tt}, (t, t')\,\mathsf{tt}) = (a, a')$ on $[\varphi]$, we define $\mathtt{glue}^*((a, a'), (t, t'))$ as the pair $(\mathtt{glue}(a, t), \mathtt{glue}(a, t)')$ where $\mathtt{glue}(a, t')\,\rho\,\rho' = \mathsf{glue}\,(a'\,\rho\,\rho')\,[\varphi \mapsto t'\,\mathsf{tt}\,\rho\,\rho']$.

3.3 Main result

One checks in a mechanical fashion that the operations we have defined above satisfy the required laws, including stability under substitution in the context (Γ, Γ'). We thus obtain the following statement.

▶ **Theorem 7** (Sconing). *Assume the parameters $\mathcal{C}, \mathbb{I}, \mathbb{F}$ satisfy the assumptions of Section 2.2. Then given any cubical cwf $\mathcal{M}$ that is size-compatible in the sense of the beginning of Section 3, the sconing $\mathcal{M}^*$ is a cubical cwf with operations defined as above. We further have a morphism $\mathcal{M}^* \to \mathcal{M}$ of cubical cwfs given by the first projection.*

4 Homotopy canonicity

We fix parameters $\mathcal{C}, \mathbb{I}, \mathbb{F}$ as before. To make our homotopy canonicity result independent of Conjecture 2 concerning initiality of the term model, we phrase it directly using the *initial model* $\mathcal{I}$, initial in the category of cubical cwfs with respect to the parameters $\mathcal{C}, \mathbb{I}, \mathbb{F}$. Its existence can be justified generically following [28, 25]. It is size-compatible in the sense of Section 3: internally, $\mathsf{Hom}_{\mathcal{I}}(\Delta, \Gamma)$ and $\mathsf{Elem}_{\mathcal{I}}(\Gamma, A)$ live in the lowest universe U_0 for all Γ, Δ, A.

▶ **Theorem 8** (Homotopy canonicity). *Assume the parameters $\mathcal{C}, \mathbb{I}, \mathbb{F}$ satisfy the assumptions of Section 2.2. In the internal language of presheaves over $\mathcal{C}$, given a closed natural $n : \mathsf{Elem}(1, \mathtt{N})$ in the initial model $\mathcal{I}$, we have a numeral $k : \mathbb{N}$ with $p : \mathsf{Elem}(1, \mathtt{Path}(\mathtt{N}, n, \mathtt{S}^k(\mathtt{0})))$.*

Proof. We start the arguing reasoning externally. Using Theorem 7, we build the sconing $\mathcal{I}^*$ of $\mathcal{I}$. Using initiality, we obtain a section F of the cubical cwf morphism $\mathcal{I}^* \to \mathcal{I}$.

Let us now proceed in the internal language. Recall the construction of Section 3.2.4 of natural numbers in $\mathcal{I}^*$. We observe that $\Sigma(n : |\mathtt{N}|).\mathbb{N}'\, n$ forms a fibrant natural number set (in the sense of Appendix B). It is thus homotopy equivalent to $\mathbb{N}$. Under this equivalence, the first projection $\Sigma(n : |\mathtt{N}|).\mathbb{N}'\, n \to |\mathtt{N}|$ implements the map sending $k : \mathbb{N}$ to $\mathtt{S}^k(\mathtt{0})$.

Inspecting the action of F on $n : \mathsf{Elem}(1, \mathtt{N})$, we obtain $n' : \mathbb{N}'\, n$. By the preceding paragraph, this corresponds to $k : \mathbb{N}$ with a path $p' : \mathbb{I} \to |\mathtt{N}|$ from n to $\mathtt{S}^k(\mathtt{0})$. Now $p = \langle\rangle(p')$ is the desired witness of homotopy canonicity. ◀

5 Extensions

5.1 Identity types

Our treatment extends to the variation of cubical cwfs that includes identity types.

Identity types in a cubical cwf denoted as in Section 1.3 consist of the following operations and laws (omitting stability under substitution), internal to presheaves over $\mathcal{C}$. Fix A in $\mathsf{Type}(\Gamma)$. Given x, y in $\mathsf{Elem}(\Gamma, A)$, we have $\mathtt{Id}(A, x, y)$ in $\mathsf{Type}(\Gamma)$, of level n if A is. Given a in $\mathsf{Elem}(\Gamma, A)$, we have $\mathtt{refl}(a)$ in $\mathsf{Elem}(\Gamma, \mathtt{Id}(A, a, a))$. Given P in $\mathsf{Type}(\Gamma.A.A\mathsf{p}.\mathtt{Id}(A\mathsf{pp}, \mathsf{qp}, \mathsf{q}))$ and d in $\mathsf{Elem}(\Gamma.A, P[\mathsf{q}, \mathsf{q}, \mathtt{refl}(q)])$ and x, y in $\mathsf{Elem}(\Gamma, A)$ and p in $\mathsf{Elem}(\Gamma, \mathtt{Id}(A, x, y))$, we have $\mathtt{J}(P, d, x, y, p)$ in $\mathsf{Elem}(\Gamma, P[x, y, p])$. We have $\mathtt{J}(P, d, a, a, \mathtt{refl}(a)) = d[a]$.

We can interpret univalent type theory in such any such cubical cwf as per Remark 1.

The standard model of Section 2.3 has identity type $\mathtt{Id}(\langle A, \mathsf{fib}_A\rangle, x, y) : \mathsf{Type}(\Gamma)$ given by $\Pi(\rho : \Gamma).\mathsf{Id}_{A\,\rho}\,(x\,\rho)\,(y\,\rho)$ using Andrew Swan's construction of Id referenced in Appendix B. We omit the evident description of the remaining operations.

To obtain homotopy canonicity in this setting, it suffices to extend the sconing construction $\mathcal{M}^*$ of Section 3 to identity types. Given $A : \mathsf{Type}(1)$ and $A' : |A| \to \mathsf{U}^{\mathsf{fib}}_{\omega}$, we define $\mathsf{Id}'_{A,A'}$ as the fibrant indexed inductive set (as per Appendix B) over $x, y : |A|$, $p : |\mathtt{Id}(A, x, y)|$, $x' : A'\, x$, $y' : A'\, y$ with constructor $\mathsf{refl}' : \Pi(a : |A|)(a' : A'\, a).\mathsf{Id}'_{A,A'}\, a\, a\, (\mathtt{refl}(a))\, a'\, a'$.

Now fix $(A, A') : \mathsf{Type}^*(\Gamma, \Gamma')$. Given $\rho : |\Gamma|$, $\rho' : \Gamma' \rho$, and elements $(x, x'), (y, y')$ of (A, A') in $\mathcal{M}^*$, we define

$$\mathtt{Id}^*((A, A'), (x, x'), (y, y')) = (\mathtt{Id}(A, x, y), \lambda\rho\,\rho'\,p.\mathsf{Id}'_{A\rho, A'\,\rho\,\rho'}\,x\rho\;y\rho\;p\,(x'\,\rho\,\rho')\,(y'\,\rho\,\rho')).$$

Given an element (a, a') of (A, A') in $\mathcal{M}^*$, we define $\mathtt{refl}^*(a, a') = (\mathtt{refl}(a), \mathtt{refl}(a)')$ where $\mathtt{refl}(a)'\,\rho\,\rho' = \mathsf{refl}'\,a\rho\,(a'\,\rho\,\rho')$. The eliminator $\mathtt{J}((C, C'), (d, d'), (x, x'), (y, y'), (p, p'))$ is defined as $(\mathtt{J}(C, d, x, y, p), \lambda\rho\,\rho'.h'\,x\rho\;y\rho\;p\rho\;(x'\,\rho\,\rho')\,(y'\,\rho\,\rho')\,(p'\,\rho\,\rho'))$ where

$$h' : \Pi(x\,y : |A|\,\rho)(p : |\mathtt{Id}(A, x, y)|\,\rho)(x' : A'\,\rho')(y' : A'\,\rho')(p' : \mathsf{Id}'_{A\rho, A'\,\rho\,\rho'}\,x\,y\,p\,x'\,y').$$
$$P'\,(\rho, x, y, p)\,(\rho', x', y, p')\,(\mathtt{J}(P\rho^{+++}, d\rho^{+}, x, y, p))$$

is given by induction on $\mathsf{Id}'_{A\rho, A'\,\rho\,\rho'}$ via $h'\,a\,a\,(\mathtt{refl}(a))\,a'\,a'\,(\mathsf{refl}'\,a\,a') = d'\,(\rho, a)\,(\rho', a')$.

5.2 Higher inductive types

Our treatment extends to higher inductive types [33], following the semantics presented in [7]. Crucially, we have fibrant *indexed* higher inductive sets in presheaves over $\mathcal{C}$ as we have what we would call fibrant *uniformly indexed* higher inductive sets in the same fashion as in [7] and fibrant identity sets [6, 23], mirroring the derivation of fibrant indexed inductive sets from fibrant uniformly indexed inductive sets and fibrant identity sets recollected in Appendix B.[5]

Let us look at the case of the *suspension* operation in a cubical cwf, where $\mathtt{Susp}(A) : \mathsf{Type}(\Gamma)$ has constructors $\mathtt{north}, \mathtt{south}$ and $\mathtt{merid}(a, i)$ for $a : A$ and $i : \mathbb{I}$ with $\mathtt{merid}(a, 0) = \mathtt{north}$ and $\mathtt{merid}(a, 1) = \mathtt{south}$.

For the sconing model of Section 3, we define for $A : \mathsf{Type}(1)$ and $A' : |A| \to \mathsf{U}^{\mathsf{fib}}_{\omega}$ the indexed higher inductive set $\mathsf{Susp}'_{A,A'}$ over $|\mathtt{Susp}(A)|$ with constructors

$$\mathsf{north}' : \mathsf{Susp}'_{A,A'}\;\mathtt{north} \qquad\qquad \mathsf{south}' : \mathsf{Susp}'_{A,A'}\;\mathtt{south}$$

$$\mathsf{merid}'\,a\,a'\,i : (\mathsf{Susp}\,A)'(\mathtt{merid}(a, i))[i = 0 \mapsto \mathsf{north}', i = 1 \mapsto \mathsf{south}']$$

for $a : |a|$ and $a' : A'\,a$ and $i : \mathbb{I}$ (using the notation of [7]). In the above translation to a uniformly indexed higher inductive set, the constructor north' will for example be replaced by $\mathsf{north}'' : \mathsf{Id}_{|\mathtt{Susp}(A)|}\,u\,\mathtt{north} \to \mathsf{Susp}'_{A,A'}\;u$.

Given $(A, A') : \mathsf{Type}^*(\Gamma, \Gamma')$, we then define $\mathtt{Susp}^*(A, A') = (\mathtt{Susp}(A), \lambda\rho\rho'.\,\mathsf{Susp}'_{A\rho, A'\,\rho\,\rho'})$, with constructors and eliminator treated as in Section 5.1.

References

1. P. Aczel. On Relating Type Theories and Set Theories. In T. Altenkirch, B. Reus, and W. Naraschewski, editors, *Types for Proofs and Programs*, volume 1657 of *Lecture Notes in Computer Science*, pages 1–18. Springer Verlag, Berlin, Heidelberg, New York, 1999.
2. C. Angiuli, G. Brunerie, T. Coquand, K.-B. Hou (Favonia), B. Harper, and D. Licata. Cartesian cubical type theory. preprint available from the home page of D. Licata, 2017.
3. Jean-Philippe Bernardy, Patrik Jansson, and Ross Paterson. Parametricity and Dependent Types. In *ICFP '10*, pages 345–356, 2010.
4. M. Bezem, T. Coquand, and S. Huber. A Model of Type Theory in Cubical Sets. In R. Matthes and A. Schubert, editors, *19th International Conference on Types for Proofs and Programs (TYPES 2013)*, volume 26 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 107–128. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2014.

[5] We stress that the use of "set" in this context refers to the types of the language of presheaves over $\mathcal{C}$, not homotopy sets.

5 John Cartmell. Generalised algebraic theories and contextual categories. *Ann. Pure Appl. Logic*, 32:209–243, 1986. doi:10.1016/0168-0072(86)90053-9.

6 C. Cohen, T. Coquand, S. Huber, and A. Mörtberg. Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom. In T. Uustalu, editor, *21st International Conference on Types for Proofs and Programs (TYPES 2015)*, volume 69 of *Leibniz International Proceedings in Informatics (LIPIcs)*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018.

7 T. Coquand, S. Huber, and A. Mörtberg. On Higher Inductive Types in Cubical Type Theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, pages 255–264, New York, NY, USA, 2018. ACM.

8 Thierry Coquand. Canonicity and normalization for dependent type theory. *Theoretical Computer Science*, 2019.

9 P. Dybjer. Internal Type Theory. In *Lecture Notes in Computer Science*, pages 120–134. Springer Verlag, Berlin, Heidelberg, New York, 1996.

10 S. Eilenberg. On the relation between the fundamental group of a space and the higher homotopy groups. *Fundamenta Mathematicae*, 32(1):169–175, 1939.

11 P. Gabriel and M. Zisman. *Calculus of fractions and homotopy theory*, volume 35 of *Ergebnisse der Mathematik und ihrer Grenzgebiete*. Springer, 1967.

12 N. Gambino and J. Kock. Polynomial functors and polynomial monads. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 154, pages 153–192. Cambridge University Press, 2013.

13 N. Gambino and C. Sattler. The Frobenius condition, right properness, and uniform fibrations. *Journal of Pure and Applied Algebra*, 221(12):3027–3068, 2017.

14 K. Gödel. Über eine bisher noch nicht benützte Erweiterung des finiten Standpunkts. *Dialectica*, 12:280–287, 1958.

15 M. Hofmann. Syntax and semantics of dependent types. In A.M. Pitts and P. Dybjer, editors, *Semantics and logics of computation*, volume 14 of *Publ. Newton Inst.*, pages 79–130. Cambridge University Press, Cambridge, 1997.

16 S. Huber. Canonicity for Cubical Type Theory. *Journal of Automated Reasoning*, 2018.

17 A. Joyal. Notes on clans and tribes, 2017. arXiv:1710.10238.

18 C. Kapulkin and P. LeFanu Lumsdaine. The simplicial model of univalent foundations (after Voevodsky). *arXiv preprint*, 2012. arXiv:1211.2851.

19 K. Kapulkin and V. Voevodsky. Cubical approach to straightening. preprint available from the home page of K. Kapulkin, 2018.

20 G.M. Kelly. A unified treatment of transfinite constructions for free algebras, free monoids, colimits, associated sheaves, and so on. *Bulletin of the Australian Mathematical Society*, 22(1):1–83, 1980.

21 D. R. Licata, I. Orton, A. M. Pitts, and B. Spitters. Internal Universes in Models of Homotopy Type Theory. In Hélène Kirchner, editor, *3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018)*, volume 108 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 22:1–22:17. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018.

22 P. LeFanu Lumsdaine and M. Shulman. Semantics of higher inductive types. *arXiv preprint*, 2017. arXiv:1705.07088.

23 I. Orton and A. M. Pitts. Axioms for Modelling Cubical Type Theory in a Topos. In *25th EACSL Annual Conference on Computer Science Logic (CSL 2016)*, volume 62 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 24:1–24:19, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

24 E. Palmgren and S.J. Vickers. Partial Horn logic and cartesian categories. *Annals of Pure and Applied Logic*, 145(3):314–353, 2007.

25 Erik Palmgren and Steven J Vickers. Partial Horn logic and cartesian categories. *Annals of Pure and Applied Logic*, 145(3):314–353, 2007.

26 E. Riehl and M. Shulman. A type theory for synthetic ∞-categories. *Higher Structures*, 1(1), 2018.

27 M. Shulman. Univalence for inverse diagrams and homotopy canonicity. *Mathematical Structures in Computer Science*, 25(5):1203–1277, 2015.

28 Jonathan Sterling. Algebraic type theory and universe hierarchies. *arXiv preprint*, 2019. `arXiv:1902.08848`.

29 T. Streicher. *Semantics of Type Theory.* Progress in Theoretical Computer Science. Birkhäuser Basel, 1991.

30 A. Swan. An Algebraic Weak Factorisation System on 01-Substitution Sets: A Constructive Proof. *Journal of Logic & Analysis*, 8(1):1–35, 2016.

31 A. Swan. Semantics of higher inductive types, 2017. On the HoTT mailing list.

32 W. W. Tait. Intensional Interpretations of Functionals of Finite Type I. *Journal of Symbolic Logic*, 32(2):198–212, 1967.

33 The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. `http://homotopytypetheory.org/book`, Institute for Advanced Study, 2013.

34 V. Voevodsky. The equivalence axiom and univalent models of type theory. (Talk at CMU on February 4, 2010). Preprint arXiv:1402.5556 [math.LO], 2014. `arXiv:1402.5556`.

A Rules of the term model

We denote the objects of our base category $\mathcal{C}$ by X, Y, Z and its morphisms by f, g, h. In the term model $\mathcal{T}$ morphisms $f\colon Y \to X$ act on judgments at stage X via an implicit substitution, while for substitutions on object variables we will use explicit substitutions. For this to make sense we first define the raw expressions as a presheaf: at stage X this is given by

$$
\begin{array}{rcl}
\Gamma, \Delta & ::= & \varepsilon \mid \Gamma.A \\
A, B, t, u, v & ::= & \mathtt{q} \mid t\sigma \mid \mathtt{U}_n \mid \Pi(A,B) \mid \mathtt{lam}(u) \mid \mathtt{app}(u,v) \\
& \mid & \Sigma(A,B) \mid \mathtt{pair}(u,v) \mid \mathtt{fst}(u) \mid \mathtt{snd}(u) \\
& \mid & \mathtt{Path}(\bar{A},u,v) \mid \langle\rangle\bar{u} \mid \mathtt{ap}(u,r) \\
& \mid & \mathtt{Glue}(A,\varphi,\bar{B},\bar{u}) \mid \mathtt{glue}(v,\bar{u}) \mid \mathtt{unglue}(u) \\
& \mid & \mathtt{fill}(\bar{A},\varphi,b,\bar{u},r) \mid \dots \\
\bar{A}, \bar{B}, \bar{u}, \bar{v} & ::= & (A_{f,r})_{f,r} \mid (A_f)_{f\in[\varphi]} \\
\sigma, \tau, \delta & ::= & \mathsf{p} \mid \mathsf{id} \mid \sigma\tau \mid (\sigma, u) \mid ()
\end{array}
$$

where $b \in \{0,1\}$, $\varphi \in \mathbb{F}(X)$, and we skipped the constants for natural numbers. Above, we have families of expressions, say $\bar{A} = (A_{f,r})_{f,r}$, whose index set ranges over certain Y, $f\colon Y \to X$, and $r \in \mathbb{I}(Y)$, and $A_{f,r}$ is a raw expression at stage Y; likewise $(A_f)_{f\in[\varphi](X)}$ consists of raw expressions A_f at stage Y for $f\colon Y \to X$ in the sieve $[\varphi]$ on X. (The exact index sets will be clear from the typing rules below.) All other occurrences of r above have $r \in \mathbb{I}(X)$. The restrictions along $f\colon Y \to X$ on the raw syntax then leave all the usual cwf structure untouched, so we have $\mathtt{q}f = \mathtt{q}$ and $(\Pi(A,B))f = \Pi(Af, Bf)$, and uses the restrictions in $\mathbb{I}$ and $\mathbb{F}$ accordingly, e.g., $(\mathtt{ap}(u,r))f = \mathtt{ap}(uf, rf)$, and we will re-index families according to $\bar{A}f = (A_{gf,rf})$ for $\bar{A} = (A_{g,r})_{g,r}$.

To get the *initial* cubical cwf we in fact need more annotations to the syntax in order to be able to define a partial interpretation (cf. [29, 15]) on the raw syntax. But to enhance readability we suppress these annotations.

We will now describe a type system indexed by stages X. The forms of judgment are:

$$
\Gamma \vdash_X \qquad \Gamma \vdash_X A \qquad \Gamma \vdash_X A = B \qquad \Gamma \vdash_X t : A \qquad \Gamma \vdash_X t = u : A \qquad \sigma\colon \Delta \to_X \Gamma
$$

where the involved expressions are at stage X.

▶ Remark 9. In cubical type theory as described in [6] we did not index judgments by objects X but allowed extending context by interval variables instead. Loosely speaking, a judgment $\Gamma \vdash_{\{i_1,\ldots,i_n\}} \mathcal{J}$ corresponds to $i_1 : \mathbb{I}, \ldots, i_n : \mathbb{I}, \Gamma \vdash \mathcal{J}$ given the setting of [6].

As mentioned above we have the rule:

$$\frac{\Gamma \vdash_X \mathcal{J} \qquad f\colon Y \to X}{\Gamma f \vdash_Y \mathcal{J} f}$$

At each stage we have all the usual rules valid in a cwf with Π-types, Σ-types, universes, and natural numbers. We will present some of the rules, but skip all congruence rules.

$$\frac{}{\varepsilon \vdash_X} \qquad \frac{\Gamma \vdash_X \qquad \Gamma \vdash_X A}{\Gamma.A \vdash_X} \qquad \frac{\Gamma \vdash_X A \qquad \sigma\colon \Delta \to_X \Gamma}{\Delta \vdash_X A\sigma} \qquad \frac{\Gamma \vdash_X t : A \qquad \sigma\colon \Delta \to_X \Gamma}{\Delta \vdash_X t\sigma : A\sigma}$$

$$\frac{\Gamma \vdash_X A}{\Gamma.A \vdash_X \mathsf{q} : A\mathsf{p}} \qquad \frac{\Gamma \vdash_X t : A \qquad \Gamma \vdash_X A = B}{\Gamma \vdash_X t : B} \qquad \frac{\Gamma \vdash_X}{\mathsf{id}\colon \Gamma \to_X \Gamma} \qquad \frac{\Gamma \vdash_X}{()\colon \Gamma \to \varepsilon}$$

$$\frac{\Gamma \vdash_X A}{\mathsf{p}\colon \Gamma.A \to_X \Gamma} \qquad \frac{\sigma\colon \Delta \to_X \Gamma \qquad \tau\colon \Theta \to_X \Delta}{\sigma\tau\colon \Theta \to_X \Gamma}$$

$$\frac{\sigma\colon \Delta \to_X \Gamma \qquad \Gamma \vdash_X A \qquad \Delta \vdash_X u : A\sigma}{(\sigma, u)\colon \Delta \to_X \Gamma.A}$$

$$\frac{\Gamma.A \vdash_X B}{\Gamma \vdash_X \Pi(A,B)} \qquad \frac{\Gamma.A \vdash_X B \qquad \Gamma.A \vdash_X b : B}{\Gamma \vdash_X \mathtt{lam}(b) : \Pi(A,B)} \qquad \frac{\Gamma \vdash_X w : \Pi(A,B) \qquad \Gamma \vdash_X u : A}{\Gamma \vdash_X \mathsf{app}(w,u) : B[u]}$$

where we write $[u]$ for (id, u) and σ^+ for $(\sigma\mathsf{p}, \mathsf{q})$. The judgmental equalities (skipping suitable premises, types, and contexts) are:

$$\mathsf{id}\,\sigma = \sigma\,\mathsf{id} = \sigma \quad (\sigma\tau)\delta = \sigma(\tau\delta) \quad ()\sigma = () \quad (\sigma,u)\delta = (\sigma\delta, u\delta) \quad \mathsf{p}(\sigma,u) = \sigma$$

$$\mathsf{q}(\sigma,u) = u \quad (\mathsf{p},\mathsf{q}) = \mathsf{id} \quad A\,\mathsf{id} = A \quad (A\sigma)\delta = A(\sigma\delta) \quad u\,\mathsf{id} = u \quad (u\sigma)\delta = u(\sigma\delta)$$

$$(\Pi(A,B))\sigma = \Pi(A\sigma, B\sigma^+) \quad (\mathtt{lam}(b))\sigma = \mathtt{lam}(b\sigma^+) \quad \mathsf{app}(w,u)\delta = \mathsf{app}(w\delta, u\delta)$$

$$\mathsf{app}(\mathtt{lam}(b), u) = b[u] \quad w = \mathtt{lam}(\mathsf{app}(w\mathsf{p}, \mathsf{q}))$$

We skip the rules for Σ-types and natural numbers as they are standard, but simply indexed with an object X as we did for Π-types. The rules for universes are:

$$\frac{\Gamma \vdash_X}{\Gamma \vdash_X \mathtt{U}_n} \qquad \frac{\Gamma \vdash_X}{\Gamma \vdash_X \mathtt{U}_n : \mathtt{U}_{n+1}} \qquad \frac{\Gamma \vdash_X A : \mathtt{U}_n}{\Gamma \vdash_X A : \mathtt{U}_{n+1}} \qquad \frac{\Gamma \vdash_X A : \mathtt{U}_n}{\Gamma \vdash_X A}$$

and we skip the rules for equality and closure under the type formers Π,Σ, natural numbers, `Path`, and `Glue`.

To state the rules for dependent path-types we introduce the following abbreviations. We write $\Gamma.\mathbb{I} \vdash_X \bar{A}$ if $\bar{A} = (A_{f,r})$ is a family indexed by Y, $f\colon Y \to X$, and $r \in \mathbb{I}(Y)$ such that

$$\Gamma f \vdash_Y A_{f,r} \text{ and } \Gamma fg \vdash_Z (A_{f,r})g = A_{fg,r}.$$

Given $\Gamma.\mathbb{I} \vdash_X \bar{A}$ we write $\Gamma.\mathbb{I} \vdash_X \bar{u} : \bar{A}$ whenever $\bar{u} = (u_{f,r})$ is a family indexed by Y, $f\colon Y \to X$, and $r \in \mathbb{I}(Y)$ such that

$$\Gamma f \vdash_Y u_{f,r} : A_{f,r} \text{ and } \Gamma fg \vdash_Z (u_{f,r})g = u_{fg,rg} : A_{fg,rg}.$$

The rules for the dependent path type are:

$$\frac{\Gamma.\mathbb{I} \vdash_X \bar{A} \qquad \Gamma \vdash_X u : A_{\mathsf{id}_X,0} \qquad \Gamma \vdash_X u : A_{\mathsf{id}_X,1}}{\Gamma \vdash_X \mathtt{Path}(\bar{A}, u, v)} \qquad \frac{\Gamma.\mathbb{I} \vdash_X \bar{A} \qquad \Gamma.\mathbb{I} \vdash_X \bar{u} : \bar{A}}{\Gamma \vdash_X \mathtt{lam}(\bar{u}) : \mathtt{Path}(\bar{A}, u_{\mathsf{id}_X,0}, u_{\mathsf{id}_X,1})}$$

$$\frac{\Gamma \vdash_X t : \mathtt{Path}(\bar{A}, u, v) \qquad r \in \mathbb{I}(X)}{\Gamma \vdash_X \mathtt{ap}(t, r) : A_{\mathsf{id}_X,r}}$$

$$\mathtt{ap}(\mathtt{lam}(\bar{u}), r) = u_{\mathsf{id},r} \quad t = \mathtt{lam}(\mathtt{ap}(tf, r)_{f,r}) \quad \mathtt{Path}(\bar{A}, u, v)\sigma = \mathtt{Path}((A_{f,r}\sigma f)_{f,r}, u\sigma, v\sigma)$$

$$(\mathtt{lam}(\bar{u}))\sigma = \mathtt{lam}((u_{f,r}\sigma f)_{f,r}) \quad (\mathtt{ap}(t, r))\sigma = \mathtt{ap}(t\sigma, r)$$

Note that in general these rules might have infinitely many premises. We get the non-dependent path type for $\Gamma \vdash_X A$ by using the family $A_{f,r} := Af$.

Given $\Gamma.\mathbb{I} \vdash_X \bar{A}$ and $b \in \{0, 1\}$ we write $\Gamma.\mathbb{I} \vdash_X^{\varphi,b} \bar{u} : \bar{A}$ for $\bar{u} = (u_{f,r})$ a family indexed over all Y, $f\colon Y \to X$, and $r \in \mathbb{I}(Y)$ such that either f is in the sieve $[\varphi]$ or $r = b$ and we have

$$\Gamma f \vdash_Y u_{f,r} : A_{f,r} \text{ and } \Gamma fg \vdash_Z (u_{f,r})g = u_{fg,rg} : A_{fg,rg}$$

for all $g\colon Z \to Y$. The rule for the filling operation is given by:

$$\frac{\Gamma.\mathbb{I} \vdash_X \bar{A} \qquad \varphi \in \mathbb{F}(X) \qquad b \in \{0, 1\} \qquad \Gamma.\mathbb{I} \vdash_X^{\varphi,b} \bar{u} : \bar{A} \qquad r \in \mathbb{I}(X)}{\Gamma \vdash_Y \mathtt{fill}(\bar{A}, \varphi, b, \bar{u}, r) : A_{\mathsf{id},r}}$$

with judgmental equality

$$\mathtt{fill}(\bar{A}, \varphi, b, \bar{u}, r) = u_{\mathsf{id},r} \text{ whenever } [\varphi] \text{ is the maximal sieve or } r = b.$$

For the glueing operation we only present the formation rule; the other rules are similar as in [6] but adapted to our setting. We write $\Gamma \vdash_X^{\varphi} \bar{B}$ if $\bar{B}$ is a family of B_f for $f\colon Y \to X$ in $[\varphi]$ with $\Gamma f \vdash_Y B_f$ which is compatible, i.e. $\Gamma fg \vdash_Z B_f g = B_{fg}$. In this case, we write likewise $\Gamma \vdash_X \bar{u} : \bar{B}$ if $\bar{u}$ is a compatible family of terms $\Gamma f \vdash_Y u_f : B_f$.

$$\frac{\Gamma \vdash_X A \qquad \varphi \in \mathbb{F}(X) \qquad \Gamma \vdash_X^{\varphi} \bar{B} \qquad \Gamma \vdash_X^{\varphi} \bar{u} : \mathtt{isEquiv}(\bar{B}, A)}{\Gamma \vdash_X \mathtt{Glue}(A, \varphi, \bar{B}, \bar{u})}$$

and the judgmental equality $\mathtt{Glue}(A, \varphi, \bar{B}, \bar{u}) = B_{\mathsf{id}}$ in case $[\varphi]$ is the maximal sieve, and an equation for substitution.

This formal system gives rise to a cubical cwf $\mathcal{T}$ as follows. First, define judgmental equality for contexts and substitutions as usual (we could also have those as primitive judgments). Next, we define presheaves Con and Hom on $\mathcal{C}$ by taking, say, $\mathsf{Con}(X)$ equivalence classes $[\Gamma]_\sim$ of Γ with $\Gamma \vdash_X$ modulo judgmental equality; restrictions are induced by the (implicit) substitution: $[\Gamma]_\sim f = [\Gamma f]_\sim$. Types $\mathsf{Type}(X, [\Gamma]_\sim)$ are equivalence classes of A with $\Gamma \vdash_X A$ modulo judgmental equality, and elements are defined similarly as equivalence classes.

For type formers in $\mathcal{T}$ let us look at path types: we have to give an element of $\mathsf{Type}(\Gamma)$ in a context (w.r.t. the internal language) $\Gamma : \mathsf{Con}, A : \mathbb{I} \to \mathsf{Type}(\Gamma), u : \mathsf{Elem}(\Gamma, A\,0), v : \mathsf{Elem}(\Gamma, A\,1)$. Unfolding the use of internal language, given $[\Gamma]_\sim \in \mathsf{Con}(X)$, a compatible family $[A_{f,r}]_\sim \in \mathsf{Type}(Y, [\Gamma]_\sim f)$ (for $f\colon Y \to X$ and $r \in \mathbb{I}(Y)$) and elements $[u]_\sim \in \mathsf{Elem}([\Gamma]_\sim, [A_{\mathsf{id},0}]_\sim)$ and $[v]_\sim \in ([\Gamma]_\sim, [A_{\mathsf{id},1}]_\sim)$, we have to give an element of $\mathsf{Type}(X, [\Gamma]_\sim)$, which we do by the formation rule for `Path`.

The remainder of the cubical cwf structure for $\mathcal{T}$ is defined in a similar manner, in fact the rules are designed to reflect the laws of cubical cwfs. We conjecture that we can follow a similar argument as in [29] to show that $\mathcal{T}$ is the *initial* cubical cwf. Given a cubical cwf $\mathcal{M}$ over $\mathcal{C}, \mathbb{I}, \mathbb{F}$ we first have to define *partial* interpretations of the raw syntax and then show that each derivable judgment has a defined interpretation in $\mathcal{M}$, and for equality judgments both sides of the equation have a defined interpretation in $\mathcal{M}$ and are equal. In an intuitionistic framework, this partial interpretation should be described as an inductively defined relation, which is shown to be functional. The partial interpretation $[\![-]\!]$ assigns meanings to raw judgments with the following signature:

$$
\begin{array}{ll}
[\![\Gamma \vdash_X]\!] & \in \mathsf{Con}_{\mathcal{M}}(X) \\
[\![\sigma : \Delta \to \Gamma]\!] & \in \mathsf{Hom}_{\mathcal{M}}(X, [\![\Delta \vdash_X]\!], [\![\Gamma \vdash_X]\!]) \\
[\![\Gamma \vdash_X A]\!] & \in \mathsf{Type}_{\mathcal{M}}(X, [\![\Gamma \vdash_X]\!]) \\
[\![\Gamma \vdash_X u : A]\!] & \in \mathsf{Elem}_{\mathcal{M}}(X, [\![\Gamma \vdash_X]\!], [\![\Gamma \vdash_X A]\!])
\end{array}
$$

where among the conditions for the interpretation on the left-hand side to be defined is that all references to the interpretation on the right-hand side are defined. This proceeds by structural induction on the raw syntax and for $[\![\Gamma \vdash_X \mathcal{J}]\!]$ to be defined we assume all the ingredients needed are already defined. E.g. for the path type $[\![\Gamma \vdash_X \mathtt{Path}(\bar{A}, u, v)]\!]$ we in particular have to assume that the assignment $f, r \mapsto [\![\Gamma f \vdash_Y A_{f,r}]\!]$ is defined and gives rise to a suitable input of $\mathtt{Path}_{\mathcal{M}}$.

B Indexed inductive sets in presheaves over $\mathcal{C}$

We work in the setting of Section 2.2 given by presheaves over $\mathcal{C}$.

Given a set I, a family A over I, a family B over $i : I$ and $a : A\,i$, and a map

$$s : \Pi(i : I)(a : A\,i).B\,i\,a \to I,$$

the *indexed inductive set* $\mathsf{W}_{I,A,B,s}$ is the initial algebra of the polynomial endofunctor [12] on the (internal) category of families over I sending a family X to the family

$$[\![I, A, B, s]\!]\,i = \Sigma(a : A\,i).\Pi(b : B\,i\,a).X(s\,i\,a\,b).$$

Its constructive justification as an operation in the internal language of the presheaf topos using inductive constructions of the metatheory is folklore (in a classical setting, one would use transfinite colimits [20]). If I, A, B are small with respect to a universe U_i with $i \in \{0, 1, \ldots, \omega\}$, then $\mathsf{W}_{I,A,B,s} : I \to \mathsf{U}_i$.

Let I, A, B now be small with respect to U_ω. Given elements of $\mathsf{Fill}(I, A)$ and $\mathsf{Fill}(\Sigma(i : I).A\,i, \lambda(i, a).B\,i\,a)$, we may use induction (i.e. the universal property of $W_{I,A,B,s}$) to derive an element of $\mathsf{Fill}(I, \mathsf{W}_{I,A,B,s})$. As in Section 2.2 for dependent products, this implies (using external reasoning) the internal statement $\mathsf{C}(\mathsf{W}_{I,A,B,s}\,i)$ for $i : I$ given $\mathsf{C}(A\,i)$ for all i and $\mathsf{C}(B\,i\,a)$ for all i, a. We then call $W_{I,A,B,s}$ a *fibrant uniformly indexed inductive set*. The qualifier *uniformly indexed* indicates that A is a fibrant family over I rather than a fibrant set with a "target" map to I that indicates the target sort of the constructor sup.

Given $A : \mathsf{U}_\omega$ with $\mathsf{Fill}(1, A)$, we may use the technique of Andew Swan [30, 23] to construct a (level preserving) identity set $\mathsf{Id}_A\,a_0\,a_1$ for $a_0, a_1 : A$ (different from the equality set $a_0 = a_1$) with $\mathsf{Fill}(A \times A, \lambda(a_0, a_1).\mathsf{Id}_A\,a_0\,a_1))$ and constructor $\mathsf{refl}_a : \mathsf{Id}_A\,a\,a$ for $a : A$ that has the usual elimination with respect to families $P : \Pi(a_0\,a_1 : A).\mathsf{Id}_A\,a_0\,a_1 \to \mathsf{U}_\omega$ that satisfy $\mathsf{Fill}(\Sigma(a_0\,a_1 : A).\mathsf{Id}_A\,a_0\,a_1, P)$. Using external reasoning as before, one has $\mathsf{C}(\mathsf{Id}_A\,a_0\,a_1)$ given

$\mathsf{C}(A)$, justifying calling $\mathsf{Id}_A\, a_0\, a_1$ a *fibrant identity set*; using (2) one has elimination with respect to families P of the previous signature with $\mathsf{C}(C\, a_0\, a_1\, p)$ for all a_0, a_1, p.

Using a folklore technique, we may use fibrant identity sets to derive *fibrant indexed inductive sets* from fibrant uniformly indexed inductive sets, by which we mean the following. Given $(I, \mathsf{fib}_I) : \mathsf{U}^{\mathsf{fib}}_\omega$, $(A, \mathsf{fib}_A) : \mathsf{U}^{\mathsf{fib}}_\omega$, $\langle B, \mathsf{fib}_B\rangle : A \to \mathsf{U}^{\mathsf{fib}}_\omega$ with maps $t : A \to I$ and $s : \Pi(a : A).B\, a \to I$, we have $\langle W_{I,A,B,s,t}, \mathsf{fib}_W\rangle : I \to \mathsf{U}^{\mathsf{fib}}_\omega$ (we omit the subscripts to W for readability), W living in U_i if I, A, B do, with

$$\sup : \Pi(a : A)(f : \Pi(b : B\, a).W\,(s\, a\, b)).W\,(t\, a).$$

Given $\langle P, \mathsf{fib}_P\rangle : \Pi(i : I).W \to \mathsf{U}^{\mathsf{fib}}_\omega$ with

$$h : \Pi(a : A)(f : \Pi(b : B\, a).W\,(s\, a\, b)).(\Pi(b : B\, a).P\,(s\, a\, b)\,(f\, b)) \to P\,(t\, a)\,(\sup\ a\, f),$$

we have $v : \Pi(i : I)(w : W\, i).P\, i\, w$ such that

$$v\,(t\, a)\,(\sup\ a\, f) = h\, a\, f\,(\lambda b.v\,(s\, a\, b)\,(f\, b).$$

Fibrant indexed inductive sets are used for the interpretation in the sconing model of natural numbers in Section 3, higher inductive types in Section 5.2, and identity types in Section 5.1. In practise, we will usually not bother to bring the fibrant indexed inductive set needed in into the above form and instead work explicitly with the more usual specification in terms of a list of constructors, each taking a certain number non-recursive and recursive arguments.[6] For convenience, we explain concretely the example needed in Section 3 of the fibrant indexed inductive set $\mathbb{N}'$ over

As an example, we construct the fibrant indexed inductive set $\mathbb{N}'$ needed in Section 3. There, we have a fibrant set $|\mathtt{N}| : \mathsf{U}_0$ (satisfying $\mathsf{C}(|\mathtt{N}|)$) with an element $\mathtt{0} : |\mathtt{N}|$ and an endofunction $\mathtt{S} : |\mathtt{N}| \to |\mathtt{N}|$. We wish to define the fibrant indexed inductive set $\mathbb{N}' : |\mathtt{N}| \to \mathsf{U}_0$ with constructors $\mathsf{0}' : \mathbb{N}'\,\mathtt{0}$ and $\mathsf{S}' : \Pi(n : |\mathtt{N}|\,\rho).\mathbb{N}'\, n \to \mathbb{N}'\,(\mathtt{S}\, n)$. We let $\mathbb{N}'$ be the uniformly indexed inductive set over $m : |\mathtt{N}|$ with constructors

$$\mathsf{0}'' : \mathsf{Id}_{|\mathtt{N}|}\, m\, \mathtt{0} \to \mathbb{N}'\, m,$$
$$\mathsf{S}'' : \Pi_{n:|\mathtt{N}|}.\mathsf{Id}_{|\mathtt{N}|}\, m\,(\mathtt{S}\, n) \to \mathbb{N}'\, n \to \mathbb{N}'\, m.$$

and define $\mathsf{0}' = \mathsf{0}''\, \mathsf{refl}_{\mathtt{0}}$ and $\mathsf{S}'\, n\, n' = \mathsf{S}''\, n\, \mathsf{refl}_{\mathtt{S}(n)}\, n'$. Fibrancy of Id ensures fibrancy of $\mathbb{N}'$ (i.e. $\mathsf{C}(\mathbb{N}'\, n)$ for $n : |\mathtt{N}|$). For elimination, we are given a fibrant family $P\, n\, n'$ for $n : |\mathtt{N}|$ and $n' : \mathbb{N}'\, n$ with $z' : P\,\mathtt{0}\,\mathsf{0}'$ and $s'\, n\, n'\, x : P\,(\mathtt{S}\, n)\,(\mathsf{S}'\, n\, n')$ for all n, n' and $x : P\, n\, n'$. We have to define $h'\, n\, n' : P\, n\, n'$ for all n, n' such that $h'\,\mathtt{0}\,\mathsf{0}' = z'$ and $h'\,(\mathtt{S}\, n)\,(\mathsf{S}'\, n\, n') = s'\, n\, n'\,(h'\, n\, n')$. We define h' by induction on the uniformly indexed inductive set $\mathbb{N}'$ and fibrant identity sets (using fibrancy of P) via defining equations

$$h'\,\mathsf{0}''\,\mathsf{refl}_{\mathtt{0}} = z,$$
$$h'\,(\mathtt{S} n)\,(\mathsf{S}''\, n\, \mathsf{refl}_{\mathtt{S}\, n}\, n') = s'\, n\, n'\,(h'\, n\, n').$$

[6] Note that the latter is really an instance of the former since our dependent sums, dependent products, and finite coproducts are extensional (satisfy universal properties). Conversely, the former is an instance of the latter with a single constructor taking a non-recursive and a recursive argument.

C Variations

C.1 Univalence as an axiom

Our treatment extends to the case where the glue types in a cubical cwf as in Section 1.3 are replaced by an operation $\mathsf{Elem}(\Gamma, \mathtt{iUnivalence}_n)$ for $\Gamma : \mathsf{Con}$ and $n \geq 0$, with $\mathtt{iUnivalence}_n$ defined in Remark 1.

To define this operation in the sconing model of Section 3, one first shows analogously to Lemmas 5 and 6 that $|{-}|$ preserves contractible types and that $(A, A') : \mathsf{Type}^*(\Gamma, \Gamma')$ is contractible exactly if A is contractible and $A'\,\rho\,\rho'\,a$ for $\rho : |\Gamma|$ and $\rho' : \Gamma'\,\rho$ where $a : |A|$ is the induced center of contraction. We have analogous statements for types of homotopy level $n \geq 0$i in $\mathcal{M}$, in which case we instead have to quantify over all $a : |A|$.

Given $(A, A') : \mathsf{Type}^*_n(\Gamma, \Gamma')$, we have show that the type $(S, S') = \Sigma^*(\mathtt{U}_n, \mathsf{Equiv}^*(\mathsf{q}, A))$ over (Γ, Γ') is contractible in $\mathcal{M}^*$. Without loss of generality, we may assume the center of contraction of univalence in $\mathcal{M}$ is given by the identity equivalence. Using the observations of the preceding paragraph, it suffices to show that $V' = S'\,\rho\,\rho'\,(\mathtt{pair}(A\rho, (\mathtt{lam}(\mathsf{q}), w)))$ is contractible for $\rho : |\Gamma|$ and $\rho' : \Gamma'\,\rho$ where w denotes the canonical witness that the identity map $\mathtt{lam}(\mathsf{q})$ on $A\rho$ is an equivalence in $\mathcal{M}$. Inhabitation is evident, and so it remains to show propositionality. By the case of the preceding paragraph for propositions, the second component of V' is a proposition, and thus we can ignore it for the current goal, which then becomes

$$\mathsf{isProp}\big(\Sigma(T' : |A| \to \mathsf{U}_n).\Pi(a : |A|).\mathsf{Equiv}(T'\,a, A'\,\rho\,\rho'\,a)\big)$$

and follows from univalence in the standard model, justified by glueing.

C.2 Canonicity

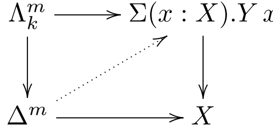
A similar sconing argument may be used to provide a reduction-free canonicity argument for cubical type theory with computation rules for filling at type formers alternative to the one of [16]. The key difference is that we now want the filling operation itself to be computable. In the sconing, we then define a type $(A, \langle A', \mathsf{fib}_{A'}\rangle) : \mathsf{Type}(\Gamma, \Gamma')$ to consist of $A : \mathsf{Type}(\Gamma)$ and $A'\,\rho\,\rho'\,a : \mathsf{U}_\omega$ as before, but with $\mathsf{fib}_{A'}\,\rho\,\rho' : \mathsf{C}(\Sigma(a : |A|\,\rho).A'\,\rho\,\rho'\,a)$ such that the first projection relates $\mathsf{fib}_{A'}\,\rho\,\rho'$ with the proof of fibrancy of $|A|\,\rho$. This is easiest formulated by defining an appropriate dependent version $\mathsf{C}' : \Pi(A : \mathsf{U}_\omega)(A' : A \to \mathsf{U}_\omega).\mathsf{C}(A) \to \mathsf{U}_\omega$ of C using the methods of Section 2.2.

D Simplicial set model

Choosing for $\mathcal{C}$ the simplex category Δ, for $\mathbb{I}$ the usual interval Δ^1 in simplicial sets, and for $\mathbb{F}$ a small copy $\Omega_{0,\mathsf{dec}}$ of the sublattice of Ω_0 of decidable sieves, we obtain a notion of cubical cwf with a simplicial notion of shape.

Assume now the law of excluded middle. The above choice of $\mathcal{C}, \mathbb{I}, \mathbb{F}$ satisfies all of the assumptions of Section 2.2 but one: the existence of a right adjoint to exponentiation with $\mathbb{I}$. However, the only place our development makes use of this assumption is in establishing (1). We will instead give a different definition of C that still satisfies (1). Then the rest of our development applies to simplicial sets.

A *Kan fibration structure* on a family $Y : X \to \mathsf{U}_\omega$ in simplicial sets consists of a choice of diagonal fillers in all commuting squares of the form



with left map a *horn inclusion*. Note that the codomains of horn inclusions are representable. It follows that the presheaf of Kan fibration structures indexed over the slice of simplicial sets over U_ω is representable. Given $[n] \in \Delta$ and $A \in (\mathsf{U}_\omega)_n$ (i.e. an ω-small presheaf on $\Delta/[n]$), we define $\mathsf{C}([n], A)$ as the set of Kan fibration structures on $A \colon \Delta^n \to \mathsf{U}_\omega$. This defines a level preserving map $\mathsf{C} \colon \mathsf{U}_\omega \to \mathsf{U}_\omega$. Then the representing object of the above presheaf is given by the first projection $\mathsf{U}_\omega^{\mathsf{fib}} \to \mathsf{U}_\omega$ where $\mathsf{U}_\omega^{\mathsf{fib}} = \Sigma(X : \mathsf{U}_\omega).\mathsf{C}(X)$ is defined as before.

Let us now verify (1). Given a simplicial set X with $Y \colon X \to \mathsf{U}_\omega$, a global element of $\mathsf{Fill}(X, Y)$ corresponds to a *uniform Kan fibration* structure on $\Sigma(x : X)\,(Y\,x) \to X$ in the sense of [13]. A uniform Kan fibration structure induces a Kan fibration structure naturally in X, giving the forward direction of (1). For the reverse direction, it suffices to give a uniform Kan fibration structure in the generic case, i.e. a global element of $\mathsf{Fill}(\mathsf{U}_\omega^{\mathsf{fib}}, \lambda(A, c).A)$. This is [13, Theorem 8.9, part (ii)] together with the fact proved in [11, Chapter IV] that Kan fibrations lift against pushout products of interval endpoint inclusions with (levelwise decidable) monomorphisms.[7]

Having verified (1), the rest of our development applies just as well to the case of simplicial sets. In particular, we obtain in the standard model $\mathcal{S}$ of Section 2.3 a version of the simplicial set model [18] of univalent type theory (using Section 5.1 for identity types).[8] As per Section 5.2, we furthermore obtain higher inductive types in the simplicial set model in a way that avoids (as suggested by Andrew Swan [31]) the pitfall of fibrant replacement failing to preserve size encountered in [22].

Seeing simplicial sets as a full subtopos of distributive lattice cubical sets as observed in [19], there is a functor from cubical cwfs with $(\mathcal{C}, \mathbb{I}, \mathbb{F}) = (\Delta, \Delta^1, \Omega_{0,\mathsf{dec}})$ to cubical cwfs where $\mathcal{C}$ is the Lawvere theory of distributive lattices, $\mathbb{I}$ is represented by the generic object, and $\mathbb{F}$ is the (small) sublattice of Ω_0 generated by distributive lattice equations. The cubical cwfs in the image of this functor satisfy a *sheaf condition*, which can be represented syntactically as an operation allowing one to e.g. uniquely glue together to a type $\Gamma \vdash_{\{i,j\}} A$ coherent families of types $\Gamma f \vdash_X A_f$ for f a map to X from the free distributive lattice on symbols $\{i, j\}$ such that $f\,i \leq f\,j$ or $f\,j \leq f\,i$ (compare also the tope logic of [26]).

Applying this functor to the simplicial set model $\mathcal{S}$ discussed above, we obtain an interpretation of distributive lattice cubical type theory (with $\mathbb{I}$ and $\mathbb{F}$ as above) in the sense of the current paper (crucially, without computation rules for filling at type formers) in simplicial sets. Thus, this cubical type theory is homotopically sound: can only derive statements which hold for standard homotopy types.

[7] This is the only place where excluded middle is used, to produce a cellular decomposition in terms of simplex boundary inclusions of such a monomorphism.

[8] Instead of Kan fibration structures, we can also work with the property of being a Kan fibration. Then C is valued in propositions and we would obtain in $\mathcal{S}$ a version of the simplicial set model in which being a type is truly just a property. However, choice would be needed to obtain (1).

Polymorphic Higher-Order Termination

Łukasz Czajka
Faculty of Informatics, TU Dortmund, Germany
http://www.mimuw.edu.pl/~lukaszcz/
lukaszcz@mimuw.edu.pl

Cynthia Kop
Institute of Computer Science, Radboud University Nijmegen, The Netherlands
https://www.cs.ru.nl/~cynthiakop/
c.kop@cs.ru.nl

Abstract

We generalise the termination method of higher-order polynomial interpretations to a setting with impredicative polymorphism. Instead of using weakly monotonic functionals, we interpret terms in a suitable extension of System $\mathtt{F}_\omega$. This enables a direct interpretation of rewrite rules which make essential use of impredicative polymorphism. In addition, our generalisation eases the applicability of the method in the non-polymorphic setting by allowing for the encoding of inductive data types. As an illustration of the potential of our method, we prove termination of a substantial fragment of full intuitionistic second-order propositional logic with permutative conversions.

2012 ACM Subject Classification Theory of computation → Rewrite systems; Theory of computation → Equational logic and rewriting; Theory of computation → Type theory

Keywords and phrases termination, polymorphism, higher-order rewriting, permutative conversions

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.12

Related Version Complete proofs for the results in this paper are available in an online appendix at https://arxiv.org/pdf/1904.09859.pdf.

1 Introduction

Termination of higher-order term rewriting systems [21, Chapter 11] has been an active area of research for several decades. One powerful method, introduced by v.d. Pol [23, 15], interprets terms into *weakly monotonic algebras*. In later work [6, 12], these algebra interpretations are specialised into *higher-order polynomial interpretations*, a generalisation of the popular – and highly automatable – technique of polynomial interpretations for first-order term rewriting.

The methods of weakly monotonic algebras and polynomial interpretation are both limited to *monomorphic* systems. In this paper, we will further generalise polynomial interpretations to a higher-order formalism with full impredicative polymorphism. This goes beyond shallow (rank-1, weak) polymorphism, where type quantifiers are effectively allowed only at the top of a type: it would be relatively easy to extend the methods to a system with shallow polymorphism since shallowly polymorphic rules can be seen as defining an infinite set of monomorphic rules. While shallow polymorphism often suffices in functional programming practice, there do exist interesting examples of rewrite systems which require higher-rank impredicative polymorphism.

For instance, in recent extensions of Haskell one may define a type of heterogeneous lists.

$$\begin{array}{ll} \mathtt{List} : * & \mathtt{foldl}_\sigma(f, a, \mathtt{nil}) \longrightarrow a \\ \mathtt{nil} : \mathtt{List} & \mathtt{foldl}_\sigma(f, a, \mathtt{cons}_\tau(x, l)) \longrightarrow \mathtt{foldl}_\sigma(f, f\tau a x, l) \\ \mathtt{cons} : \forall\alpha.\alpha \to \mathtt{List} \to \mathtt{List} & \\ \mathtt{foldl} : \forall\beta.(\forall\alpha.\beta \to \alpha \to \beta) \to \beta \to \mathtt{List} \to \beta & \end{array}$$

© Łukasz Czajka and Cynthia Kop;
licensed under Creative Commons License CC-BY

4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019).
Editor: Herman Geuvers; Article No. 12; pp. 12:1–12:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The above states that List is a type ($*$), gives the types of its two constructors `nil` and `cons`, and defines the corresponding fold-left function `foldl`. Each element of a heterogeneous list may have a different type. In practice, one would constrain the type variable α with a type class to guarantee the existence of some operations on list elements. The function argument of `foldl` receives the element together with its type. The $\forall$-quantifier binds type variables: a term of type $\forall\alpha.\tau$ takes a type ρ as an argument and the result is a term of type $\tau[\alpha := \rho]$.

Impredicativity of polymorphism means that the type itself may be substituted for its own type variable, e.g., if $\mathtt{f} : \forall\alpha.\tau$ then $f(\forall\alpha.\tau) : \tau[\alpha := \forall\alpha.\tau]$. Negative occurrences of impredicative type quantifiers prevent a translation into an infinite set of simply typed rules by instantiating the type variables. The above example is not directly reducible to shallow polymorphism as used in the ML programming language.

Related work. The term rewriting literature has various examples of higher-order term rewriting systems with some forms of polymorphism. To start, there are several studies that consider shallow polymorphic rewriting (e.g., [9, 11, 25]), where (as in ML-like languages) systems like `foldl` above cannot be handled. Other works consider extensions of the $\lambda\Pi$-calculus [2, 4] or the calculus of constructions [1, 26] with rewriting rules; only the latter includes full impredicative polymorphism. The termination techniques presented for these systems are mostly syntactic (e.g., a recursive path ordering [11, 26], or general schema [1]), as opposed to our more semantic method based on interpretations. An exception is [4], which defines interpretations into Π-algebras; this technique bears some similarity to ours, although the methodologies are quite different. A categorical definition for a general polymorphic rewriting framework is presented in [5], but no termination methods are considered for it.

Our approach. The technique we develop in this paper operates on *Polymorphic Functional Systems (PFSs)*, a form of higher-order term rewriting systems with full impredicative polymorphism (Section 3), that various systems of interest can be encoded into (including the example of heterogeneous fold above). Then, our methodology follows a standard procedure:

- we define a well-ordered set $(\mathcal{I}, \succ, \succeq)$ (Section 4);
- we provide a general methodology to map each PFS term s to a natural number $[\![s]\!]$, parameterised by a core interpretation for each function symbol (Section 5);
- we present a number of lemmas to make it easy to prove that $s \succ t$ or $s \succeq t$ whenever s reduces to t (Section 6).

Due to the additional complications of full polymorphism, we have elected to only generalise higher-order polynomial interpretations, and not v.d. Pol's weakly monotonic algebras. That is, terms of base type are always interpreted to natural numbers and all functions are interpreted to combinations of addition and multiplication.

We will use the system of heterogeneous fold above as a running example to demonstrate our method. However, termination of this system can be shown in other ways (e.g., an encoding in System F). Hence, we will also study a more complex example in Section 7: termination of a substantial fragment of IPC2, i.e., full intuitionistic second-order propositional logic with permutative conversions. Permutative conversions [22, Chapter 6] are used in proof theory to obtain "good" normal forms of natural deduction proofs, which satisfy e.g. the subformula property. Termination proofs for systems with permutative conversions are notoriously tedious and difficult, with some incorrect claims in the literature and no uniform methodology. It is our goal to make such termination proofs substantially easier in the future.

2 Preliminaries

In this section we recall the definition of System $\mathtt{F}_\omega$ (see e.g. [17, Section 11.7]), which will form a basis both of our interpretations and of a general syntactic framework for the investigated systems. In comparison to System F, System $\mathtt{F}_\omega$ includes type constructors which results in a more uniform treatment. We assume familiarity with core notions of lambda calculi such as substitution and α-conversion.

▶ **Definition 2.1.** Kinds *are defined inductively:* $*$ *is a kind, and if* κ_1, κ_2 *are kinds then so is* $\kappa_1 \Rightarrow \kappa_2$*. We assume an infinite set* $\mathcal{V}_\kappa$ *of* type constructor variables *of each kind* κ*. Variables of kind* $*$ *are* type variables*. We assume a fixed set* Σ^T_κ *of* type constructor symbols *paired with a kind* κ*, denoted* $c : \kappa$*. We define the set* $\mathcal{T}_\kappa$ *of* type constructors *of kind* κ *by the following grammar. Type constructors of kind* $*$ *are* types*.*

$$\begin{aligned} \mathcal{T}_* &::= \mathcal{V}_* \mid \Sigma^T_* \mid \mathcal{T}_{\kappa\Rightarrow *}\mathcal{T}_\kappa \mid \forall\mathcal{V}_\kappa\mathcal{T}_* \mid \mathcal{T}_* \to \mathcal{T}_* \\ \mathcal{T}_{\kappa_1\Rightarrow\kappa_2} &::= \mathcal{V}_{\kappa_1\Rightarrow\kappa_2} \mid \Sigma^T_{\kappa_1\Rightarrow\kappa_2} \mid \mathcal{T}_{\kappa\Rightarrow(\kappa_1\Rightarrow\kappa_2)}\mathcal{T}_\kappa \mid \lambda\mathcal{V}_{\kappa_1}\mathcal{T}_{\kappa_2} \end{aligned}$$

We use the standard notations $\forall\alpha.\tau$ *and* $\lambda\alpha.\tau$*. When* α *is of kind* κ *then we use the notation* $\forall\alpha : \kappa.\tau$*. If not indicated otherwise, we assume* α *to be a type variable. We treat type constructors up to* α*-conversion.*

▶ **Example 2.2.** If $\Sigma^T_* = \{\mathtt{List}\}$ and $\Sigma^T_{*\Rightarrow*\Rightarrow*} = \{\mathtt{Pair}\}$, types are for instance $\mathtt{List}$ and $\forall\alpha.\mathtt{Pair}\,\alpha\,\mathtt{List}$. The expression $\mathtt{Pair}\,\mathtt{List}$ is a type constructor, but not a type. If $\Sigma^T_{(*\Rightarrow*)\Rightarrow*} = \{\exists\}$ and $\sigma \in \mathcal{T}_{*\Rightarrow*}$, then both $\exists(\sigma)$ and $\exists(\lambda\alpha.\sigma\alpha)$ are types.

The compatible closure of the rule $(\lambda\alpha.\varphi)\psi \to \varphi[\alpha := \psi]$ *defines* β*-reduction on type constructors. As type constructors are (essentially) simply-typed lambda-terms, their* β*-reduction terminates and is confluent; hence every type constructor* τ *has a unique* β*-normal form* $\mathrm{nf}_\beta(\tau)$*. A* type atom *is a type in* β*-normal form which is neither an arrow* $\tau_1 \to \tau_2$ *nor a quantification* $\forall\alpha.\tau$*.*

We define $\mathrm{FTV}(\varphi)$ *– the set of free type constructor variables of the type constructor* φ *– in an obvious way by induction on* φ*. A type constructor* φ *is* closed *if* $\mathrm{FTV}(\varphi) = \emptyset$*.*

We assume a fixed type symbol $\chi_* \in \Sigma^T_*$*. For* $\kappa = \kappa_1 \Rightarrow \kappa_2$ *we define* $\chi_\kappa = \lambda\alpha : \kappa_1.\chi_{\kappa_2}$*.*

▶ **Definition 2.3.** *We assume given an infinite set* $\mathcal{V}$ *of variables, each paired with a type, denoted* $x : \tau$*. We assume given a fixed set* Σ *of* function symbols*, each paired with a closed type, denoted* $\mathtt{f} : \tau$*. Every variable* x *and every function symbol* $\mathtt{f}$ *occurs only with one type declaration.*

The set of preterms *consists of all expressions* s *such that* $s : \sigma$ *can be inferred for some type* σ *by the following clauses:*

- $x : \sigma$ *for* $(x : \sigma) \in \mathcal{V}$*.*
- $\mathtt{f} : \sigma$ *for all* $(\mathtt{f} : \sigma) \in \Sigma$*.*
- $\lambda x : \sigma.s : \sigma \to \tau$ *if* $(x : \sigma) \in \mathcal{V}$ *and* $s : \tau$*.*
- $(\Lambda\alpha : \kappa.s) : (\forall\alpha : \kappa.\sigma)$ *if* $s : \sigma$ *and* α *does not occur free in the type of a free variable of* s*.*
- $s \cdot t : \tau$ *if* $s : \sigma \to \tau$ *and* $t : \sigma$
- $s * \tau : \sigma[\alpha := \tau]$ *if* $s : \forall\alpha : \kappa.\sigma$ *and* τ *is a type constructor of kind* κ*,*
- $s : \tau$ *if* $s : \tau'$ *and* $\tau =_\beta \tau'$*.*

The set of free variables of a preterm t*, denoted* $\mathrm{FV}(t)$*, is defined in the expected way. Analogously, we define the set* $\mathrm{FTV}(t)$ *of type constructor variables occurring free in* t*. If* α *is a type then we use the notation* $\Lambda\alpha.t$*. We denote an occurrence of a variable* x *of type* τ *by* x^τ*, e.g.* $\lambda x : \tau \to \sigma.x^{\tau\to\sigma}y^\tau$*. When clear or irrelevant, we omit the type annotations,*

denoting the above term by $\lambda x.xy$. Type substitution is defined in the expected way except that it needs to change the types of variables. Formally, a type substitution changes the types associated to variables in $\mathcal{V}$. We define the equivalence relation $\equiv$ by: $s \equiv t$ iff s and t are identical modulo β-conversion in types.

Note that we present terms in orthodox Church-style, i.e., instead of using contexts each variable has a globally fixed type associated to it.

▶ **Lemma 2.4.** *If $s : \tau$ and $s \equiv t$ then $t : \tau$.*

Proof. Induction on s. ◀

▶ **Definition 2.5.** *The set of* terms *is the set of the equivalence classes of $\equiv$.*

Because β-reduction on types is confluent and terminating, every term has a canonical preterm representative – the one with all types occurring in it β-normalised. We define $\mathrm{FTV}(t)$ as the value of FTV on the canonical representative of t. We say that t is *closed* if both $\mathrm{FTV}(t) = \emptyset$ and $\mathrm{FV}(t) = \emptyset$. Because typing and term formation operations (abstraction, application, ...) are invariant under $\equiv$, we may denote terms by their (canonical) representatives and informally treat them interchangeably.

We will often abuse notation to omit $\cdot$ and $*$. Thus, st can refer to both $s \cdot t$ and $s * t$. This is not ambiguous due to typing. When writing $\sigma[\alpha := \tau]$ we implicitly assume that α and τ have the same kind. Analogously with $t[x := s]$.

▶ **Lemma 2.6** (Substitution lemma).
1. *If $s : \tau$ and $x : \sigma$ and $t : \sigma$ then $s[x := t] : \tau$.*
2. *If $t : \sigma$ then $t[\alpha := \tau] : \sigma[\alpha := \tau]$.*

Proof. Induction on the typing derivation. ◀

▶ **Lemma 2.7** (Generation lemma). *If $t : \sigma$ then there is a type σ' such that $\sigma' =_\beta \sigma$ and $\mathrm{FTV}(\sigma') \subseteq \mathrm{FTV}(t)$ and one of the following holds.*
- *$t \equiv x$ is a variable with $(x : \sigma') \in \mathcal{V}$.*
- *$t \equiv \mathtt{f}$ is a function symbol with $\mathtt{f} : \sigma'$ in Σ.*
- *$t \equiv \lambda x : \tau_1.s$ and $\sigma' = \tau_1 \to \tau_2$ and $s : \tau_2$.*
- *$t \equiv \Lambda\alpha : \kappa.s$ and $\sigma' = \forall\alpha : \kappa.\tau$ and $s : \tau$ and α does not occur free in the type of a free variable of s.*
- *$t \equiv t_1 \cdot t_2$ and $t_1 : \tau \to \sigma'$ and $t_2 : \tau$ and $\mathrm{FTV}(\tau) \subseteq \mathrm{FTV}(t)$.*
- *$t \equiv s * \tau$ and $\sigma' = \rho[\alpha := \tau]$ and $s : \forall(\alpha : \kappa).\rho$ and τ is a type constructor of kind κ.*

Proof. By analysing the derivation $t : \sigma$. To ensure $\mathrm{FTV}(\sigma') \subseteq \mathrm{FTV}(t)$, note that if $\alpha \notin \mathrm{FTV}(t)$ is of kind κ and $t : \sigma'$, then $t : \sigma'[\alpha := \chi_\kappa]$ by the substitution lemma (thus we can eliminate α). ◀

3 Polymorphic Functional Systems

In this section, we present a form of higher-order term rewriting systems based on $\mathtt{F}_\omega$: *Polymorphic Functional Systems (PFSs).* Systems of interest, such as logic systems like ICP2 and higher-order TRSs with shallow or full polymorphism can be encoded into PFSs, and then proved terminating with the technique we will develop in Sections 4 – 6.

▶ **Definition 3.1.** Kinds*,* type constructors *and* types *are defined like in Definition 2.1, parameterised by a fixed set* $\Sigma^T = \bigcup_\kappa \Sigma^T_\kappa$ *of type constructor symbols.*

Let Σ *be a set of function symbols such that for* $\mathtt{f} : \sigma \in \Sigma$*:*

$$\sigma = \forall(\alpha_1 : \kappa_1) \dots \forall(\alpha_n : \kappa_n).\sigma_1 \to \dots \to \sigma_k \to \tau \qquad (\textit{with } \tau \textit{ a type atom})$$

We define PFS terms *as in Definition 2.5 (based on Definition 2.3), parameterised by* Σ*, with the restriction that for any subterm* $s \cdot u$ *of a term* t*, we have* $s = \mathtt{f}\rho_1 \dots \rho_n u_1 \dots u_m$ *where:*

$$\mathtt{f} : \forall(\alpha_1 : \kappa_1) \dots \forall(\alpha_n : \kappa_n).\sigma_1 \to \dots \to \sigma_k \to \tau \qquad (\textit{with } \tau \textit{ a type atom and } k > m)$$

This definition does not allow for a variable or abstraction to occur at the head of an application, nor can we have terms of the form $s \cdot t * \tau \cdot q$ (although terms of the form $s \cdot t * \tau$, or $x * \tau$ with x a variable, *are* allowed to occur). To stress this restriction, we will use the notation $\mathtt{f}_{\rho_1,\dots,\rho_n}(s_1, \dots, s_m)$ as an alternative way to denote $\mathtt{f}\rho_1 \dots \rho_n s_1 \dots s_m$ when $\mathtt{f} : \forall(\alpha_1 : \kappa_1) \dots \forall(\alpha_n : \kappa_n).\sigma_1 \to \dots \to \sigma_k \to \tau$ is a function symbol in Σ with τ a type atom and $m \leq k$. This allows us to represent terms in a "functional" way, where application does not explicitly occur (only implicitly in the construction of $\mathtt{f}_{\rho_1,\dots,\rho_n}(s_1, \dots, s_m)$).

The following result follows easily by induction on term structure:

▶ **Lemma 3.2.** *If* t, s *are PFS terms then so is* $t[x := s]$.

PFS terms will be rewritten through a reduction relation $\longrightarrow_{\mathcal{R}}$ based on a (usually infinite) set of rewrite rules. To define this relation, we need two additional notions.

▶ **Definition 3.3.** *A* replacement *is a function* $\delta = \gamma \circ \omega$ *satisfying:*

1. ω *is a type constructor substitution,*
2. γ *is a term substitution such that* $\gamma(\omega(x)) : \omega(\tau)$ *for every* $(x : \tau) \in \mathcal{V}$.

For τ *a type constructor, we use* $\delta(\tau)$ *to denote* $\omega(\tau)$*. We use the notation* $\delta[x := t] = \gamma[x := t] \circ \omega$*. Note that if* $t : \tau$ *then* $\delta(t) : \delta(\tau)$*.*

▶ **Definition 3.4.** *A* σ-context C_σ *is a PFS term with a fresh function symbol* $\Box_\sigma \notin \Sigma$ *of type* σ *occurring exactly once. By* $C_\sigma[t]$ *we denote a PFS term obtained from* C_σ *by substituting* t *for* $\Box_\sigma$*. We drop the* σ *subscripts when clear or irrelevant.*

Now, the rewrite rules are simply a set of term pairs, whose monotonic closure generates the rewrite relation.

▶ **Definition 3.5.** *A set* $\mathcal{R}$ *of term pairs* (ℓ, r) *is a set of* rewrite rules *if: (a)* $\mathrm{FV}(r) \subseteq \mathrm{FV}(\ell)$*; (b)* ℓ *and* r *have the same type; and (c) if* $(\ell, r) \in \mathcal{R}$ *then* $(\delta(\ell), \delta(r)) \in \mathcal{R}$ *for any replacement* δ*. The reduction relation* $\longrightarrow_{\mathcal{R}}$ *on PFS terms is defined by:*

$$t \longrightarrow_{\mathcal{R}} s \textit{ iff } t = C[\ell] \textit{ and } s = C[r] \textit{ for some } (\ell, r) \in \mathcal{R} \textit{ and context } C.$$

▶ **Definition 3.6.** *A* Polymorphic Functional System (PFS) *is a triple* $(\Sigma^T, \Sigma, \mathcal{R})$ *where* Σ^T *is a set of type constructor symbols,* Σ *a set of function symbols (restricted as in Def. 3.1), and* $\mathcal{R}$ *is a set of rules as in Definition 3.5. A term of a PFS* A *is referred to as an* A*-term.*

While PFS-terms are a restriction from the general terms of system $\mathtt{F}_\omega$, the reduction relation allows us to actually encode, e.g., system $\mathtt{F}$ as a PFS: we can do so by including the symbol $@ : \forall\alpha\forall\beta.(\alpha \to \beta) \to \alpha \to \beta$ in Σ and adding all rules of the form $@_{\sigma,\tau}(\lambda x.s, t) \longrightarrow s[x := t]$. Similarly, β-reduction of type abstraction can be modelled by including a symbol

$\mathtt{A} : \forall\alpha : * \Rightarrow *.\forall\beta.(\forall\gamma.\alpha\gamma) \to \alpha\beta$ and rules $\mathtt{A}_{\lambda\gamma.\sigma,\tau}(\Lambda\gamma.s) \longrightarrow s[\gamma := \tau]$.[1] We can also use rules $(\Lambda\alpha.s) * \tau \longrightarrow s[\alpha := \tau]$ without the extra symbol, but to apply our method it may be convenient to use the extra symbol, as it creates more liberty in choosing an interpretation.

▶ **Example 3.7** (Fold on heterogenous lists). The example from the introduction may be represented as a PFS with one type symbol $\mathtt{List} : *$, the following function symbols:

$$\begin{array}{rcl} @ & : & \forall\alpha\forall\beta.(\alpha \to \beta) \to \alpha \to \beta \\ \mathtt{A} & : & \forall\alpha : * \Rightarrow *.\forall\beta.(\forall\gamma.\alpha\gamma) \to \alpha\beta \\ \mathtt{nil} & : & \mathtt{List} \\ \mathtt{cons} & : & \forall\alpha.\alpha \to \mathtt{List} \to \mathtt{List} \\ \mathtt{foldl} & : & \forall\beta.(\forall\alpha.\beta \to \alpha \to \beta) \to \beta \to \mathtt{List} \to \beta \end{array}$$

and the following rules (which formally represents an infinite set of rules: one rule for each choice of types σ, τ and PFS terms s, t, etc.):

$$\begin{array}{rcl} @_{\sigma,\tau}(\lambda x : \sigma.s, t) & \longrightarrow & s[x := t] \\ \mathtt{A}_{\lambda\alpha.\sigma,\tau}(\Lambda\alpha.s) & \longrightarrow & s[\alpha := \tau] \\ \mathtt{foldl}_\sigma(f, s, \mathtt{nil}) & \longrightarrow & s \\ \mathtt{foldl}_\sigma(f, s, \mathtt{cons}_\tau(h, t)) & \longrightarrow & \mathtt{foldl}_\sigma(f, @_{\tau,\sigma}(@_{\sigma,\tau\to\sigma}(\mathtt{A}_{\lambda\alpha.\sigma\to\alpha\to\sigma,\tau}(f), s), h), t) \end{array}$$

4 A well-ordered set of interpretation terms

In polynomial interpretations of first-order term rewriting [21, Chapter 6.2], each term s is mapped to a natural number $[\![s]\!]$, such that $[\![s]\!] > [\![t]\!]$ whenever $s \longrightarrow_{\mathcal{R}} t$. In higher-order rewriting, this is not practical; instead, following [15], terms are mapped to weakly monotonic functionals according to their type (i.e., terms with a 0-order type are mapped to natural numbers, terms with a 1-order type to weakly monotonic functions over natural numbers, terms with a 2-order type to weakly monotonic functionals taking weakly monotonic functions as arguments, and so on). In this paper, to account for full polymorphism, we will interpret PFS terms to a set $\mathcal{I}$ of *interpretation terms* in a specific extension of System $\mathtt{F}_\omega$. This set is defined in Section 4.1; we provide a well-founded partial ordering $\succ$ on $\mathcal{I}$ in Section 4.2.

Although our world of interpretation terms is quite different from the weakly monotonic functionals of [15], there are many similarities. Most pertinently, every interpretation term $\lambda x.s$ essentially defines a weakly monotonic function from $\mathcal{I}$ to $\mathcal{I}$. This, and the use of both addition and multiplication in the definition of $\mathcal{I}$, makes it possible to lift higher-order polynomial interpretations [6] to our setting. We prove weak monotonicity in Section 4.3.

4.1 Interpretation terms

▶ **Definition 4.1.** *The set $\mathcal{Y}$ of* interpretation types *is the set of types as in Definition 2.1 with $\Sigma^T = \{\mathtt{nat} : *\}$, i.e., there is a single type constant $\mathtt{nat}$. Then $\chi_* = \mathtt{nat}$.*

The set $\mathcal{I}$ of interpretation terms *is the set of terms from Definition 2.5 (see also Definition 2.3) where as types we take the interpretation types and for the set Σ of function symbols we take $\Sigma = \{n : \mathtt{nat} \mid n \in \mathbb{N}\} \cup \Sigma_f$, where $\Sigma_f = \{\oplus : \forall\alpha.\alpha \to \alpha \to \alpha, \otimes : \forall\alpha.\alpha \to \alpha \to \alpha, \mathtt{flatten} : \forall\alpha.\alpha \to \mathtt{nat}, \mathtt{lift} : \forall\alpha.\mathtt{nat} \to \alpha\}$.*

[1] The use of a type constructor variable α of kind $* \Rightarrow *$ makes it possible to do type substitution as part of a rule. An application $s * \tau$ with $s : \forall\gamma.\sigma$ is encoded as $\mathtt{A}_{\lambda\gamma.\sigma,\tau}(s)$, so α is substituted with $\lambda\gamma.\tau$. This is well-typed because $(\lambda\gamma.\sigma)\gamma =_\beta \sigma$ and $(\lambda\gamma.\sigma)\tau =_\beta \sigma[\gamma := \tau]$.

For easier presentation, we write $\oplus_\tau$, $\otimes_\tau$, etc., instead of $\oplus * \tau$, $\otimes * \tau$, etc. We will also use $\oplus$ and $\otimes$ in *infix, left-associative* notation, and omit the type denotation where it is clear from context. Thus, $s \oplus t \oplus u$ should be read as $\oplus_\sigma (\oplus_\sigma s\, t)\, u$ if s has type σ. Thus, our interpretation terms include natural numbers with the operations of addition and multiplication. It would not cause any fundamental problems to add more monotonic operations, e.g., exponentiation, but we refrain from doing so for the sake of simplicity.

Normalising interpretation terms

The set $\mathcal{I}$ of interpretation terms can be reduced through a relation $\rightsquigarrow$, that we will define below. This relation will be a powerful aid in defining the partial ordering $\succ$ in Section 4.2.

▶ **Definition 4.2.** *We define the relation $\rightsquigarrow$ on interpretation terms as the smallest relation on $\mathcal{I}$ for which the following properties are satisfied:*

1. *if $s \rightsquigarrow t$ then both $\lambda x.s \rightsquigarrow \lambda x.t$ and $\Lambda\alpha.s \rightsquigarrow \Lambda\alpha.t$*
2. *if $s \rightsquigarrow t$ then $u \cdot s \rightsquigarrow u \cdot t$*
3. *if $s \rightsquigarrow t$ then both $s \cdot u \rightsquigarrow t \cdot u$ and $s * \sigma \rightsquigarrow t * \sigma$*
4. $(\lambda x : \sigma.s) \cdot t \rightsquigarrow s[x := t]$ *and* $(\Lambda\alpha.s) * \sigma \rightsquigarrow s[\alpha := \sigma]$ *(β-reduction)*
5. $\oplus_{\mathtt{nat}} \cdot n \cdot m \rightsquigarrow n + m$ *and* $\otimes_{\mathtt{nat}} \cdot n \cdot m \rightsquigarrow n \times m$
6. $\circ_{\sigma\to\tau} \cdot s \cdot t \rightsquigarrow \lambda x : \sigma.\, \circ_\tau \cdot (s \cdot x) \cdot (t \cdot x)$ *for* $\circ \in \{\oplus, \otimes\}$
7. $\circ_{\forall\alpha.\sigma} \cdot s \cdot t \rightsquigarrow \Lambda\alpha.\, \circ_\sigma \cdot (s * \alpha) \cdot (t * \alpha)$ *for* $\circ \in \{\oplus, \otimes\}$
8. $\mathtt{flatten}_{\mathtt{nat}} \cdot s \rightsquigarrow s$
9. $\mathtt{flatten}_{\sigma\to\tau} \cdot s \rightsquigarrow \mathtt{flatten}_\tau \cdot (s \cdot (\mathtt{lift}_\sigma \cdot 0))$
10. $\mathtt{flatten}_{\forall\alpha:\kappa.\sigma} \cdot s \rightsquigarrow \mathtt{flatten}_{\sigma[\alpha:=\chi_\kappa]} \cdot (s * \chi_\kappa)$
11. $\mathtt{lift}_{\mathtt{nat}} \cdot s \rightsquigarrow s$
12. $\mathtt{lift}_{\sigma\to\tau} \cdot s \rightsquigarrow \lambda x : \sigma.\mathtt{lift}_\tau \cdot s$
13. $\mathtt{lift}_{\forall\alpha.\sigma} \cdot s \rightsquigarrow \Lambda\alpha.\mathtt{lift}_\sigma \cdot s$

Recall Definition 2.5 and Definition 4.1 of the set of interpretation terms $\mathcal{I}$ as the set of the equivalence classes of $\equiv$. So, for instance, $\mathtt{lift}_{\mathtt{nat}}$ above denotes the equivalence class of all preterms $\mathtt{lift}_\sigma$ with $\sigma =_\beta \mathtt{nat}$. Hence, the above rules are invariant under $\equiv$ (by confluence of β-reduction on types), and they correctly define a relation on interpretation terms. We say that s is a redex *if s reduces by one of the rules 4–13. A* final interpretation term *is an interpretation term $s \in \mathcal{I}$ such that (a) s is closed, and (b) s is in normal form with respect to $\rightsquigarrow$. We let $\mathcal{I}^f$ be the set of all final interpretation terms. By $\mathcal{I}_\tau$ ($\mathcal{I}^f_\tau$) we denote the set of all (final) interpretation terms of interpretation type τ.*

An important difference with System $\mathtt{F}_\omega$ and related ones is that the rules for $\oplus_\tau$, $\otimes_\tau$, $\mathtt{flatten}_\tau$ and $\mathtt{lift}_\tau$ depend on the type τ. In particular, type substitution in terms may create redexes. For instance, if α is a type variable then $\oplus_\alpha t_1 t_2$ is not a redex, but $\oplus_{\sigma\to\tau} t_1 t_2$ is. This makes the question of termination subtle. Indeed, System $\mathtt{F}_\omega$ is extremely sensitive to modifications which are not of a logical nature. For instance, adding a constant $\mathtt{J} : \forall\alpha\beta.\alpha \to \beta$ with a reduction rule $\mathtt{J}\tau\tau \rightsquigarrow \lambda x : \tau.x$ makes the system non-terminating [8]. This rule breaks parametricity by making it possible to compare two arbitrary types. Our rules do not allow such a definition. Moreover, the natural number constants cannot be distinguished "inside" the system. In other words, we could replace all natural number constants with 0 and this would not change the reduction behaviour of terms. So for the purposes of termination, the type $\mathtt{nat}$ is essentially a singleton. This implies that, while we have polymorphic functions between an arbitrary type α and $\mathtt{nat}$ which are not constant when seen "from outside" the system, they are constant for the purposes of reduction "inside" the system (as they would have to be in a parametric $\mathtt{F}_\omega$-like system). Intuitively, these properties of our system ensure that it stays "close enough" to $\mathtt{F}_\omega$ so that the standard termination proof still generalises.

Now we state some properties of $\leadsto$, including strong normalisation. Because of space limitations, most (complete) proofs are delegated to [3, Appendix A.1].

▶ **Lemma 4.3** (Subject reduction). *If $t : \tau$ and $t \leadsto t'$ then $t' : \tau$.*

Proof. By induction on the definition of $t \leadsto t'$, using Lemmas 2.6 and 2.7. ◀

▶ **Theorem 4.4.** *If $t : \sigma$ then t is terminating with respect to $\leadsto$.*

Proof. By an adaptation of the Tait-Girard computability method. The proof is an adaptation of chapters 6 and 14 from the book [7], and chapters 10 and 11 from the book [17]. Details are available in [3, Appendix A.1]. ◀

▶ **Lemma 4.5.** *Every term $s \in \mathcal{I}$ has a unique normal form $s\downarrow$. If s is closed then so is $s\downarrow$.*

Proof. One easily checks that $\leadsto$ is locally confluent. Since the relation is terminating by Theorem 4.4, it is confluent by Newman's lemma. ◀

▶ **Lemma 4.6.** *The only final interpretation terms of type* `nat` *are the natural numbers.*

▶ **Example 4.7.** Let $s \in \mathcal{I}_{\mathtt{nat}\to\mathtt{nat}}$ and $t \in \mathcal{I}_{\mathtt{nat}}$. Then we can reduce $(s \oplus \mathtt{lift}_{\mathtt{nat}\to\mathtt{nat}}(1)) \cdot t \leadsto (\lambda x.sx \oplus \mathtt{lift}_{\mathtt{nat}\to\mathtt{nat}}(1)x) \cdot t \leadsto st \oplus \mathtt{lift}_{\mathtt{nat}\to\mathtt{nat}}(1)t \leadsto st \oplus (\lambda y.\mathtt{lift}_{\mathtt{nat}}(1))t \leadsto st \oplus \mathtt{lift}_{\mathtt{nat}}(1) \leadsto st \oplus 1$. If s and t are variables, this term is in normal form.

4.2 The ordering pair $(\succeq, \succ)$

With these ingredients, we are ready to define the well-founded partial ordering $\succ$ on $\mathcal{I}$. In fact, we will do more: rather than a single partial ordering, we will define an *ordering pair*: a pair of a quasi-ordering $\succeq$ and a compatible well-founded ordering $\succ$. The quasi-ordering $\succeq$ often makes it easier to prove $s \succ t$, since it suffices to show that $s \succeq s' \succ t' \succeq t$ for some interpretation terms s', t'. Having $\succeq$ will also allow us to use rule removal (Theorem 6.1).

▶ **Definition 4.8.** *Let $R \in \{\succ^0, \succeq^0\}$. For* closed *$s, t \in \mathcal{I}_\sigma$ and closed σ in β-normal form, the relation $s \; R_\sigma \; t$ is defined coinductively by the following rules.*

$$\frac{s\downarrow \; R \; t\downarrow \text{ in } \mathbb{N}}{s \; R_{\mathtt{nat}} \; t} \qquad \frac{s \cdot q \; R_\tau \; t \cdot q \text{ for all } q \in \mathcal{I}^f_\sigma}{s \; R_{\sigma\to\tau} \; t} \qquad \frac{s * \tau \; R_{\mathrm{nf}_\beta(\sigma[\alpha:=\tau])} \; t * \tau \text{ for all closed } \tau \in \mathcal{T}_\kappa}{s \; R_{\forall(\alpha:\kappa).\sigma} \; t}$$

We define $s \approx^0_\sigma t$ if both $s \succeq^0_\sigma t$ and $t \succeq^0_\sigma s$. We drop the type subscripts when clear or irrelevant.

Note that in the case for `nat` the terms $s\downarrow$, $t\downarrow$ are natural numbers by Lemma 4.6 ($s\downarrow, t\downarrow$ are closed and in normal form, so they are final interpretation terms).

Intuitively, the above definition means that e.g. $s \succ^0 t$ iff there exists a possibly infinite derivation tree using the above rules. In such a derivation tree all leaves must witness $s\downarrow > t\downarrow$ in natural numbers. However, this also allows for infinite branches, which solves the problem of repeating types due to impredicative polymorphism. If e.g. $s \succ^0_{\forall\alpha.\alpha} t$ then $s * \forall\alpha.\alpha \succ^0_{\forall\alpha.\alpha} t * \forall\alpha.\alpha$, which forces an infinite branch in the derivation tree. According to our definition, any infinite branch may essentially be ignored.

Formally, the above coinductive definition of e.g. $\succ^0_\sigma$ may be interpreted as defining the largest relation such that if $s \succ^0_\sigma t$ then:

- $\sigma = \mathtt{nat}$ and $s\downarrow > t\downarrow$ in $\mathbb{N}$, or
- $\sigma = \tau_1 \to \tau_2$ and $s \cdot q \succ^0_{\tau_2} t \cdot q$ for all $q \in \mathcal{I}^f_{\tau_1}$, or
- $\sigma = \forall(\alpha : \kappa).\rho$ and $s * \tau \succ^0_{\mathrm{nf}_\beta(\rho[\alpha:=\tau])} t * \tau$ for all closed $\tau \in \mathcal{T}_\kappa$.

For more background on coinduction see e.g. [14, 16, 10]. In this paper we use a few simple coinductive proofs to establish the basic properties of $\succ$ and $\succeq$. Later, we just use these properties and the details of the definition do not matter.

▶ **Definition 4.9.** *A* closure $\mathcal{C} = \gamma \circ \omega$ *is a replacement such that $\omega(\alpha)$ is closed for each type constructor variable α, and $\gamma(x)$ is closed for each term variable x. For arbitrary types σ and arbitrary terms $s, t \in \mathcal{I}$ we define $s \succ_\sigma t$ if for every closure $\mathcal{C}$ we can obtain $\mathcal{C}(s) \succ^c_{\mathrm{nf}_\beta(\mathcal{C}(\sigma))} \mathcal{C}(t)$ coinductively with the above rules. The relations $\succeq_\sigma$ and $\approx_\sigma$ are defined analogously.*

Note that for closed s, t and closed σ in β-normal form, $s \succ_\sigma t$ iff $s \succ^0_\sigma t$ (and analogously for $\succeq, \approx$). In this case we shall often omit the superscript 0.

The definition of $\succ$ and $\succeq$ may be reformulated as follows.

▶ **Lemma 4.10.** *$t \succeq s$ if and only if for every closure $\mathcal{C}$ and every sequence $u_1, \ldots, u_n$ of closed terms and closed type constructors such that $\mathcal{C}(t)u_1 \ldots u_n : \mathtt{nat}$ we have $(\mathcal{C}(t)u_1 \ldots u_n){\downarrow} \geq (\mathcal{C}(s)u_1 \ldots u_n){\downarrow}$ in natural numbers. An analogous result holds with $\succ$ or $\approx$ instead of $\succeq$.*

Proof. The direction from left to right follows by induction on n; the other by coinduction. ◀

In what follows, all proofs by coinduction could be reformulated to instead use the lemma above. However, this would arguably make the proofs less perspicuous. Moreover, a coinductive definition is better suited for a formalisation – the coinductive proofs here could be written in Coq almost verbatim.

Our next task is to show that $\succeq$ and $\succ$ have the desired properties of an ordering pair; e.g., transitivity and compatibility. We first state a simple lemma that will be used implicitly.

▶ **Lemma 4.11.** *If $\tau \in \mathcal{Y}$ is closed and β-normal, then $\tau = \mathtt{nat}$ or $\tau = \tau_1 \to \tau_2$ or $\tau = \forall \alpha \sigma$.*

▶ **Lemma 4.12.** *$\succ$ is well-founded.*

Proof. It suffices to show this for closed terms and closed types in β-normal form, because any infinite sequence $t_1 \succ_\tau t_2 \succ_\tau t_3 \succ_\tau \ldots$ induces an infinite sequence $\mathcal{C}(t_1) \succ_{\mathrm{nf}_\beta(\mathcal{C}(\tau))} \mathcal{C}(t_2) \succ_{\mathrm{nf}_\beta(\mathcal{C}(\tau))} \mathcal{C}(t_3) \succ_{\mathrm{nf}_\beta(\mathcal{C}(\tau))} \ldots$ for any closure $\mathcal{C}$. By induction on the size of a β-normal type τ (with size measured as the number of occurrences of $\forall$ and $\to$) one proves that there does not exist an infinite sequence $t_1 \succ_\tau t_2 \succ_\tau t_3 \succ_\tau \ldots$ For instance, if α has kind κ and $t_1 \succ_{\forall \alpha \tau} t_2 \succ_{\forall \alpha \tau} t_3 \succ_{\forall \alpha \tau} \ldots$ then $t_1 * \chi_\kappa \succ_{\tau'} t_2 * \chi_\kappa \succ_{\tau'} t_3 * \chi_\kappa \succ_{\tau'} \ldots$, where $\tau' = \mathrm{nf}_\beta(\tau[\alpha := \chi_\kappa])$. Because τ is in β-normal form, all redexes in $\tau[\alpha := \chi_\kappa]$ are created by the substitution and must have the form $\chi_\kappa u$. Hence, by the definition of χ_κ (see Definition 2.1) the type τ' is smaller than τ. This contradicts the inductive hypothesis. ◀

▶ **Lemma 4.13.** *Both $\succ$ and $\succeq$ are transitive.*

Proof. We show this for $\succ$, the proof for $\succeq$ being analogous. Again, it suffices to prove this for closed terms and closed types in β-normal form. We proceed by coinduction.

If $t_1 \succ_{\mathtt{nat}} t_2 \succ_{\mathtt{nat}} t_3$ then $t_1{\downarrow} > t_2{\downarrow} > t_3{\downarrow}$, so $t_1{\downarrow} > t_3{\downarrow}$. Thus $t_1 \succ_{\mathtt{nat}} t_3$.

If $t_1 \succ_{\sigma \to \tau} t_2 \succ_{\sigma \to \tau} t_3$ then $t_1 \cdot q \succ_\tau t_2 \cdot q \succ_\tau t_3 \cdot q$ for $q \in \mathcal{I}^f_\sigma$. Hence $t_1 \cdot q \succ_\tau t_3 \cdot q$ for $q \in \mathcal{I}^f_\sigma$ by the coinductive hypothesis. Thus $t_1 \succ_{\sigma \to \tau} t_3$.

If $t_1 \succ_{\forall (\alpha:\kappa)\sigma} t_2 \succ_{\forall (\alpha:\kappa)\sigma} t_3$ then $t_1 * \tau \succ_{\sigma'} t_2 * \tau \succ_{\sigma'} t_3 * \tau$ for any closed τ of kind κ, where $\sigma' = \mathrm{nf}_\beta(\sigma[\alpha := \tau])$. By the coinductive hypothesis $t_1 * \tau \succ_{\sigma'} t_3 * \tau$; thus $t_1 \succ_{\forall \alpha \sigma} t_3$. ◀

▶ **Lemma 4.14.** *$\succeq$ is reflexive.*

Proof. By coinduction one shows that $\succeq_\sigma$ is reflexive on closed terms for closed β-normal σ. The case of $\succeq$ is then immediate from definitions. ◀

▶ **Lemma 4.15.** *The relations $\succeq$ and $\succ$ are compatible, i.e., $\succ \cdot \succeq \subseteq \succ$ and $\succeq \cdot \succ \subseteq \succ$.*

Proof. By coinduction, analogous to the transitivity proof. ◀

▶ **Lemma 4.16.** *If $t \succ s$ then $t \succeq s$.*

Proof. By coinduction. ◀

▶ **Lemma 4.17.** *If $t \rightsquigarrow s$ then $t \approx s$.*

Proof. Follows from Lemma 4.10, noting that $t \rightsquigarrow s$ implies $\mathcal{C}(t) \rightsquigarrow \mathcal{C}(s)$ for all closures $\mathcal{C}$. ◀

▶ **Lemma 4.18.** *Assume $t \succ s$ (resp. $t \succeq s$). If $t \rightsquigarrow t'$ or $t' \rightsquigarrow t$ then $t' \succ s$ (resp. $t' \succeq s$). If $s \rightsquigarrow s'$ or $s' \rightsquigarrow s$ then $t \succ s'$ (resp. $t \succeq s'$).*

Proof. Follows from Lemma 4.17, transitivity and compatibility. ◀

▶ **Corollary 4.19.** *For $R \in \{\succ, \succeq, \approx\}$: $s\ R\ t$ if and only if $s\downarrow\ R\ t\downarrow$.*

▶ **Example 4.20.** We can prove that $x \oplus \mathtt{lift}_{\mathtt{nat}\to\mathtt{nat}}(1) \succ x$: by definition, this holds if $s \oplus \mathtt{lift}_{\mathtt{nat}\to\mathtt{nat}}(1) \succ s$ for all closed s, so if $(s \oplus \mathtt{lift}_{\mathtt{nat}\to\mathtt{nat}}(1))u \succ su$ for all closed s, u. Following Example 4.7 and Lemma 4.18, this holds if $su \oplus 1 \succ su$. By definition, this is the case if $(su \oplus 1)\downarrow > (su)\downarrow$ in the natural numbers, which clearly holds for any s, u.

4.3 Weak monotonicity

We will now show that $s \succeq s'$ implies $t[x := s] \succeq t[x := s']$ (weak monotonicity). For this purpose, we prove a few lemmas, many of which also apply to $\succ$, stating the preservation of $\succeq$ under term formation operations. We will need these results in the next section.

▶ **Lemma 4.21.** *For $R \in \{\succeq, \succ\}$: if $t\,R\,s$ then $tu\,R\,su$ with u a term or type constructor.*

Proof. Follows from definitions. ◀

▶ **Lemma 4.22.** *For $R \in \{\succeq, \succ\}$: if $n\,R\,m$ then $\mathtt{lift}_\sigma n\,R\,\mathtt{lift}_\sigma m$ for all types σ.*

Proof. Without loss of generality we may assume σ closed and in β-normal form. By coinduction we show $\mathtt{lift}(n)u_1 \dots u_k \succeq \mathtt{lift}(m)u_1 \dots u_k$ for closed $u_1, \dots, u_k$. First note that $(\mathtt{lift}\,t)u_1 \dots u_k \rightsquigarrow^* \mathtt{lift}(t)$ (with a different type subscript in $\mathtt{lift}$ on the right side, omitted for conciseness). If $\sigma = \mathtt{nat}$ then $(\mathtt{lift}(n)u_1 \dots u_k)\downarrow = n \geq m = (\mathtt{lift}(m)u_1 \dots u_k)\downarrow$. If $\sigma = \tau_1 \to \tau_2$ then by the coinductive hypothesis $\mathtt{lift}(n)u_1 \dots u_k q \succeq_{\tau_2} \mathtt{lift}(m)u_1 \dots u_k q$ for any $q \in \mathcal{I}^f_{\tau_1}$, so $\mathtt{lift}(n)u_1 \dots u_k \succeq_\sigma \mathtt{lift}(m)u_1 \dots u_k$ by definition. If $\sigma = \forall(\alpha : \kappa)\tau$ then by the coinductive hypothesis $\mathtt{lift}(n)u_1 \dots u_k\xi \succeq_{\sigma'} \mathtt{lift}(m)u_1 \dots u_k\xi$ for any closed $\xi \in \mathcal{T}_\kappa$, where $\sigma' = \tau[\alpha := \xi]$. Hence $\mathtt{lift}(n)u_1 \dots u_k \succeq_\sigma \mathtt{lift}(m)u_1 \dots u_k$ by definition. ◀

▶ **Lemma 4.23.** *For $R \in \{\succeq, \succ\}$: if $t\,R_\sigma\,s$ then $\mathtt{flatten}_\sigma t\,R_{\mathtt{nat}}\,\mathtt{flatten}_\sigma s$ for all types σ.*

Proof. Without loss of generality we may assume σ is closed and in β-normal form. Using Lemma 4.18, the lemma follows by induction on σ. ◀

▶ **Lemma 4.24.** *For $R \in \{\succeq, \succ\}$: if $t\,R\,s$ then $\lambda x.t\,R\,\lambda x.s$ and $\Lambda\alpha.t\,R\,\Lambda\alpha.s$.*

Proof. Assume $t \succeq_\tau s$ and $x : \sigma$. Let $\mathcal{C}$ be a closure. We need to show $\mathcal{C}(\lambda x.t) \succeq_{\mathcal{C}(\sigma\to\tau)} \mathcal{C}(\lambda x.s)$. Let $u \in \mathcal{I}^f_{\mathcal{C}(\sigma)}$. Then $\mathcal{C}' = \mathcal{C}[x := u]$ is a closure and $\mathcal{C}'(t) \succeq_{\mathcal{C}(\tau)} \mathcal{C}'(s)$. Hence $\mathcal{C}(t)[x := u] \succeq_{\mathcal{C}(\tau)} \mathcal{C}(s)[x := u]$. By Lemma 4.18 this implies $\mathcal{C}(\lambda x.t)u \succeq_{\mathcal{C}(\tau)} \mathcal{C}(\lambda x.s)u$. Therefore $\mathcal{C}(\lambda x.t) \succeq_{\mathcal{C}(\sigma\to\tau)} \mathcal{C}(\lambda x.s)$. The proof for $\succ$ is analogous. ◀

▶ **Lemma 4.25.** *Let s, t, u be terms of type σ.*

1. *If $s \succeq t$ then $s \oplus_\sigma u \succeq t \oplus_\sigma u$, $u \oplus_\sigma s \succeq u \oplus_\sigma t$, $s \otimes_\sigma u \succeq t \otimes_\sigma u$, and $u \otimes_\sigma s \succeq u \otimes_\sigma t$.*
2. *If $s \succ t$ then $s \oplus_\sigma u \succ t \oplus_\sigma u$ and $u \oplus_\sigma s \succ u \oplus_\sigma t$. Moreover, if additionally $u \succeq \mathtt{lift}_\sigma(1)$ then also $s \otimes_\sigma u \succ t \otimes_\sigma u$ and $u \otimes_\sigma s \succ u \otimes_\sigma t$.*

Proof. It suffices to prove this for closed s, t, u and closed σ in β-normal form. The proof is similar to the proof of Lemma 4.22. For instance, we show by coinduction that for closed $w_1, \dots, w_n$ (denoted $\vec{w}$): if $s\vec{w} \succ t\vec{w}$ and $u\vec{w} \succeq \mathtt{lift}(1)\vec{w}$ then $(s \otimes u)\vec{w} \succ (t \otimes u)\vec{w}$. ◀

The following lemma depends on the lemmas above. The full proof may be found in [3, Appendix A.2]. The proof is actually quite complex, and uses a method similar to Girard's method of candidates for the termination proof.

▶ **Lemma 4.26** (Weak monotonicity)**.** *If $s \succeq s'$ then $t[x := s] \succeq t[x := s']$.*

▶ **Corollary 4.27.** *If $s \succeq s'$ then $ts \succeq ts'$.*

5 A reduction pair for PFS terms

Recall that our goal is to prove termination of reduction in a PFS. To do so, in this section we will define a systematic way to generate *reduction pairs*. We fix a PFS A, and define:

▶ **Definition 5.1.** *A binary relation R on A-terms is* monotonic *if $R(s,t)$ implies $R(C[s], C[t])$ for every context C (we assume s, t have the same type σ).*

A reduction pair *is a pair $(\succeq^A, \succ^A)$ of a quasi-order $\succeq^A$ on A-terms and a well-founded ordering $\succ^A$ on A-terms such that: (a) $\succeq^A$ and $\succ^A$ are compatible, i.e., $\succ^A \cdot \succeq^A \subseteq \succ^A$ and $\succeq^A \cdot \succ^A \subseteq \succ^A$, and (b) $\succeq^A$ and $\succ^A$ are both monotonic.*

If we can generate such a pair with $\ell \succ^A r$ for each rule $(\ell, r) \in \mathcal{R}$, then we easily see that the PFS A is terminating. (If we merely have $\ell \succ^A r$ for *some* rules and $\ell \succeq^A r$ for the rest, we can still progress with the termination proof, as we will discuss in Section 6.) To generate this pair, we will define the notion of an *interpretation* from the set of A-terms to the set $\mathcal{I}$ of interpretation terms, and thus lift the ordering pair $(\succeq, \succ)$ to A. In the next section, we will show how this reduction pair can be used in practice to prove termination of PFSs.

One of the core ingredients of our interpretation function is a mapping to translate types:

▶ **Definition 5.2.** *A* type constructor mapping *is a function $\mathcal{TM}$ which maps each type constructor symbol to a closed interpretation type constructor of the same kind. A fixed type constructor mapping $\mathcal{TM}$ is extended inductively to a function from type constructors to closed interpretation type constructors in the expected way. We denote the extended* interpretation (type) mapping *by $[\![\sigma]\!]$. Thus, e.g. $[\![\forall\alpha.\sigma]\!] = \forall\alpha.[\![\sigma]\!]$ and $[\![\sigma \to \tau]\!] = [\![\sigma]\!] \to [\![\tau]\!]$.*

▶ **Lemma 5.3.** $[\![\sigma]\!][\alpha := [\![\tau]\!]] = [\![\sigma[\alpha := \tau]]\!]$

Proof. Induction on σ. ◀

Similarly, we employ a *symbol mapping* as the key ingredient to interpret PFS terms.

▶ **Definition 5.4.** *Given a fixed type constructor mapping $\mathcal{TM}$, a* symbol mapping *is a function $\mathcal{J}$ which assigns to each function symbol $\mathtt{f} : \rho$ a closed interpretation term $\mathcal{J}(\mathtt{f})$ of type $[\![\rho]\!]$. For a fixed symbol mapping $\mathcal{J}$, we define the* interpretation mapping $[\![s]\!]$ *inductively:*

$$\begin{array}{rclrclrcl} [\![x]\!] & = & x & [\![\Lambda\alpha.s]\!] & = & \Lambda\alpha.[\![s]\!] & [\![t_1 \cdot t_2]\!] & = & [\![t_1]\!] \cdot [\![t_2]\!] \\ [\![\mathtt{f}]\!] & = & \mathcal{J}(\mathtt{f}) & [\![\lambda x : \sigma.s]\!] & = & \lambda x : [\![\sigma]\!].[\![s]\!] & [\![t * \tau]\!] & = & [\![t]\!] * [\![\tau]\!] \end{array}$$

Note that $[\![\sigma]\!], [\![\tau]\!]$ above depend on $\mathcal{TM}$. Essentially, $[\![\cdot]\!]$ substitutes $\mathcal{TM}(\mathtt{c})$ for type constructor symbols $\mathtt{c}$, and $\mathcal{J}(\mathtt{f})$ for function symbols $\mathtt{f}$, thus mapping A-terms to interpretation terms. This translation preserves typing:

▶ **Lemma 5.5.** *If $s : \sigma$ then $[\![s]\!] : [\![\sigma]\!]$.*

Proof. By induction on the form of s, using Lemma 5.3. ◀

▶ **Lemma 5.6.** *For all s, t, x, α, τ: $[\![s]\!][\alpha := [\![\tau]\!]] = [\![s[\alpha := \tau]]\!]$ and $[\![s]\!][x := [\![t]\!]] = [\![s[x := t]]\!]$.*

Proof. Induction on s. ◀

▶ **Definition 5.7.** *For a fixed type constructor mapping $\mathcal{TM}$ and symbol mapping $\mathcal{J}$, the* interpretation pair $(\succeq^{\mathcal{J}}, \succ^{\mathcal{J}})$ *is defined as follows: $s \succeq^{\mathcal{J}} t$ if $[\![s]\!] \succeq [\![t]\!]$, and $s \succ^{\mathcal{J}} t$ if $[\![s]\!] \succ [\![t]\!]$.*

▶ Remark 5.8. The polymorphic lambda-calculus has a much greater expressive power than the simply-typed lambda-calculus. Inductive data types may be encoded, along with their constructors and recursors with appropriate derived reduction rules. This makes our interpretation method easier to apply, even in the non-polymorphic setting, thanks to more sophisticated "programming" in the interpretations. The reader is advised to consult e.g. [7, Chapter 11] for more background and explanations. We demonstrate the idea by presenting an encoding for the recursive type `List` and its fold-left function (see also Ex. 5.14).

▶ **Example 5.9.** Towards a termination proof of Example 3.7, we set $\mathcal{TM}(\mathtt{List}) = \forall\beta.(\forall\alpha.\beta \to \alpha \to \beta) \to \beta \to \beta$ and $\mathcal{J}(\mathtt{nil}) = \Lambda\beta.\lambda f : \forall\alpha.\beta \to \alpha \to \beta.\lambda x : \beta.x$. If we additionally choose $\mathcal{J}(\mathtt{foldl}) = \Lambda\beta.\lambda f.\lambda x.\lambda l.l\beta f x \oplus \mathtt{lift}_\beta(1)$, we have $[\![\mathtt{foldl}_\sigma(f, s, \mathtt{nil})]\!] = (\Lambda\beta.\lambda f.\lambda x.\lambda l.l\beta f x \oplus \mathtt{lift}_\beta(1))[\![\sigma]\!] f s(\Lambda\beta.\lambda f.\lambda x.x) \rightsquigarrow^* s \oplus \mathtt{lift}_{[\![\sigma]\!]}(1)$ by β-reduction steps. An extension of the proof from Example 4.20 shows that this term $\succ [\![s]\!]$.

It is easy to see that $\succeq^{\mathcal{J}}$ and $\succ^{\mathcal{J}}$ have desirable properties such as transitivity, reflexivity (for $\succeq^{\mathcal{J}}$) and well-foundedness (for $\succ^{\mathcal{J}}$). However, $\succ^{\mathcal{J}}$ is not necessarily monotonic. Using the interpretation from Example 5.9, $[\![\mathtt{foldl}_\sigma(\lambda x.s, t, \mathtt{nil})]\!] = [\![\mathtt{fold}_\sigma(\lambda x.w, t, \mathtt{nil})]\!]$ regardless of s and w, so a reduction in s would not cause a decrease in $\succ^{\mathcal{J}}$. To obtain a reduction pair, we must impose certain conditions on $\mathcal{J}$; in particular, we will require that $\mathcal{J}$ is *safe*.

▶ **Definition 5.10.** *If $s_1 \succ s_2$ implies $t[x := s_1] \succ t[x := s_2]$, then the interpretation term t is* safe *for x. A symbol mapping $\mathcal{J}$ is* safe *if for all $\mathtt{f} : \forall(\alpha_1 : \kappa_1) \dots \forall(\alpha_n : \kappa_n).\sigma_1 \to \dots \to \sigma_k \to \tau$ with τ a type atom we have: $\mathcal{J}(\mathtt{f}) = \Lambda\alpha_1 \dots \alpha_n.\lambda x_1 \dots x_k.t$ with t safe for each x_i.*

▶ **Lemma 5.11.**
1. *$x u_1 \dots u_m$ is safe for x.*
2. *If t is safe for x then so are $\mathtt{lift}(t)$ and $\mathtt{flatten}(t)$.*
3. *If s_1 is safe for x or s_2 is safe for x then $s_1 \oplus s_2$ is safe for x.*
4. *If either (a) s_1 is safe for x and $s_2 \succeq \mathtt{lift}(1)$, or (b) s_2 is safe for x and $s_1 \succeq \mathtt{lift}(1)$, then $s_1 \otimes s_2$ is safe for x.*
5. *If t is safe for x then so is $\Lambda\alpha.t$ and $\lambda y.t$ ($y \neq x$).*

Proof. Each point follows from one of the lemmas proven before, Lemma 4.16, Lemma 4.26, Lemma 4.15 and the transitivity of $\succeq$. For instance, for the first, assume $s_1 \succ s_2$ and let $u_i^j = u_i[x := s_j]$. Then $(xu_1 \dots u_m)[x := s_1] = s_1 u_1^1 \dots u_m^1$. By Lemma 4.21 we have $s_1 u_1^1 \dots u_m^1 \succ s_2 u_1^1 \dots u_m^1$. By Lemma 4.16 and Lemma 4.26 we have $u_i^1 \succeq u_i^2$. By Corollary 4.27 and the transitivity of $\succeq$ we obtain $s_2 u_1^1 \dots u_m^1 \succeq s_2 u_1^2 \dots u_m^2$. By Lemma 4.15 finally $(xu_1 \dots u_m)[x := s_1] = s_1 u_1^1 \dots u_m^1 \succ s_2 u_1^2 \dots u_m^2 = (xu_1 \dots u_m)[x := s_2]$. ◀

▶ **Lemma 5.12.** *If $\mathcal{J}$ is safe then $\succ^{\mathcal{J}}$ is monotonic.*

Proof. Assume $s_1 \succ^{\mathcal{J}} s_2$. By induction on a context C we show $C[s_1] \succ^{\mathcal{J}} C[s_2]$. If $C = \Box$ then this is obvious. If $C = \lambda x.C'$ or $C = \Lambda\alpha.C'$ then $C'[s_1] \succ^{\mathcal{J}} C'[s_2]$ by the inductive hypothesis, and thus $C[s_1] \succ^{\mathcal{J}} C[s_2]$ follows from Lemma 4.24 and definitions. If $C = C't$ then $C'[s_1] \succ^{\mathcal{J}} C'[s_2]$ by the inductive hypothesis, so $C[s_1] \succ^{\mathcal{J}} C[s_2]$ follows from definitions.

Finally, assume $C = t \cdot C'$. Then $t = \mathtt{f}\rho_1 \dots \rho_n t_1 \dots t_m$ where $\mathtt{f} : \forall(\alpha_1 : \kappa_1) \dots \forall(\alpha_n : \kappa_n).\sigma_1 \to \dots \to \sigma_k \to \tau$ with τ a type atom, $m < k$, and $\mathcal{J}(\mathtt{f}) = \Lambda\alpha_1 \dots \alpha_n.\lambda x_1 \dots x_k.u$ with u safe for each x_i. Without loss of generality assume $m = k - 1$. Then $[\![C[s_i]]\!] \leadsto u'[x_k := [\![C'[s_i]]\!]]$ where $u' = u[\alpha_1 := [\![\rho_1]\!]] \dots [\alpha_n := [\![\rho_n]\!]][x_1 := [\![t_1]\!]] \dots [x_{k-1} := [\![t_{k-1}]\!]]$. By the inductive hypothesis $[\![C'[s_1]]\!] \succ [\![C'[s_2]]\!]$. Hence $u'[x_k := [\![C'[s_1]]\!]] \succ u'[x_k := [\![C'[s_2]]\!]]$, because u is safe for x_k. Thus $[\![C[s_1]]\!] \succ [\![C[s_2]]\!]$ by Lemma 4.18. ◀

▶ **Theorem 5.13.** *If $\mathcal{J}$ is safe then $(\succeq^{\mathcal{J}}, \succ^{\mathcal{J}})$ is a reduction pair.*

Proof. By Lemmas 4.13 and 4.14, $\succeq^{\mathcal{J}}$ is a quasi-order. Lemmas 4.12 and 4.13 imply that $\succ^{\mathcal{J}}$ is a well-founded ordering. Compatibility follows from Lemma 4.15. Monotonicity of $\succeq^{\mathcal{J}}$ follows from Lemma 4.26. Monotonicity of $\succ^{\mathcal{J}}$ follows from Lemma 5.12. ◀

▶ **Example 5.14.** The following is a safe interpretation for the PFS from Example 3.7:

$$
\begin{array}{rcll}
\mathcal{TM}(\mathtt{List}) & = & \forall\beta.(\forall\alpha.\beta \to \alpha \to \beta) \to \beta \to \beta & \\
\mathcal{J}(@) & = & \Lambda\alpha.\Lambda\beta.\lambda f.\lambda x. & f \cdot x \oplus \mathtt{lift}_\beta(\mathtt{flatten}_\alpha(x)) \\
\mathcal{J}(\mathtt{A}) & = & \Lambda\alpha.\Lambda\beta.\lambda x. & x * \beta \\
\mathcal{J}(\mathtt{nil}) & = & & \Lambda\beta.\lambda f.\lambda x.x \\
\mathcal{J}(\mathtt{cons}) & = & \Lambda\alpha.\lambda h.\lambda t. & \Lambda\beta.\lambda f.\lambda x.t\beta f(f\alpha x h \oplus \mathtt{lift}_\beta(\mathtt{flatten}_\beta(x) \oplus \\
& & & \qquad \mathtt{flatten}_\alpha(h))) \oplus \\
& & & \qquad \mathtt{lift}_\beta(\mathtt{flatten}_\beta(f\alpha x h) \oplus \mathtt{flatten}_\alpha(h) \oplus 1) \\
\mathcal{J}(\mathtt{foldl}) & = & \Lambda\beta.\lambda f.\lambda x.\lambda l. & l\beta f x \oplus \mathtt{lift}_\beta(\mathtt{flatten}_{\forall\alpha.\beta\to\alpha\to\beta}(f) \oplus \\
& & & \qquad \mathtt{flatten}_\beta(x) \oplus 1)
\end{array}
$$

Note that $\mathcal{J}(\mathtt{cons})$ is *not* required to be safe for x, since x is not an argument of `cons`: following its declaration, `cons` takes one type and two terms as arguments. The variable x is only part of the *interpretation*. Note also that the current interpretation is a mostly straightforward extension of Example 5.9: we retain the same *core* interpretations (which, intuitively, encode @ and `A` as forms of application and encode a list as the function that executes a fold over the list's contents), but we add a clause $\oplus\mathtt{lift}(\mathtt{flatten}(x))$ for each argument x that the initial interpretation is not safe for. The only further change is that, in $\mathcal{J}(\mathtt{cons})$, the part between brackets has to be extended. This was necessitated by the change to $\mathcal{J}(\mathtt{foldl})$, in order for the rules to still be oriented (as we will do in Example 6.6).

6 Proving termination with rule removal

A PFS A is certainly terminating if its reduction relation $\longrightarrow_{\mathcal{R}}$ is contained in a well-founded relation, which holds if $\ell \succ^{\mathcal{J}} r$ for all its rules (ℓ, r). However, sometimes it is cumbersome to find an interpretation that orients all rules strictly. To illustrate, the interpretation of Example 5.14 gives $\ell \succ^{\mathcal{J}} r$ for two of the rules and $\ell \succeq^{\mathcal{J}} r$ for the others (as we will see in Example 6.6). In such cases, proof progress is still achieved through *rule removal*.

▶ **Theorem 6.1.** *Let $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$, and suppose that $\mathcal{R}_1 \subseteq \succ^{\mathcal{R}}$ and $\mathcal{R}_2 \subseteq \succeq^{\mathcal{R}}$ for a reduction pair $(\succeq^{\mathcal{R}}, \succ^{\mathcal{R}})$. Then $\longrightarrow_{\mathcal{R}}$ is terminating if and only if $\longrightarrow_{\mathcal{R}_2}$ is (so certainly if $\mathcal{R}_2 = \emptyset$).*

Proof. Monotonicity of $\succeq^{\mathcal{R}}$ and $\succ^{\mathcal{R}}$ implies that $\longrightarrow_{\mathcal{R}_1} \subseteq \succ^{\mathcal{R}}$ and $\longrightarrow_{\mathcal{R}_2} \subseteq \succeq^{\mathcal{R}}$.

By well-foundedness of $\succ^{\mathcal{R}}$, compatibility of $\succeq^{\mathcal{R}}$ and $\succ^{\mathcal{R}}$, and transitivity of $\succeq^{\mathcal{R}}$, every infinite $\longrightarrow_{\mathcal{R}}$ sequence can contain only finitely many $\longrightarrow_{\mathcal{R}_1}$ steps. ◀

The above theorem gives rise to the following *rule removal* algorithm:

1. While $\mathcal{R}$ is non-empty:
   a. Construct a reduction pair $(\succeq^{\mathcal{R}}, \succ^{\mathcal{R}})$ such that all rules in $\mathcal{R}$ are oriented by $\succeq^{\mathcal{R}}$ or $\succ^{\mathcal{R}}$, and at least one of them is oriented using $\succ^{\mathcal{R}}$.
   b. Remove all rules ordered by $\succ^{\mathcal{R}}$ from $\mathcal{R}$.

If this algorithm succeeds, we have proven termination.

To use this algorithm with the pair $(\succeq^{\mathcal{J}}, \succ^{\mathcal{J}})$ from Section 5, we should identify an interpretation $(\mathcal{TM}, \mathcal{J})$ such that (a) $\mathcal{J}$ is safe, (b) all rules can be oriented with $\succeq^{\mathcal{J}}$ or $\succ^{\mathcal{J}}$, and (c) at least one rule is oriented with $\succ^{\mathcal{J}}$. The first requirement guarantees that $(\succeq^{\mathcal{J}}, \succ^{\mathcal{J}})$ is a reduction pair (by Theorem 5.13). Lemma 5.11 provides some sufficient safety criteria. The second and third requirements have to be verified for each individual rule.

▶ **Example 6.2.** We continue with our example of fold on heterogeneous lists. We prove termination by rule removal, using the symbol mapping from Example 5.14. We will show:

$$\begin{array}{rcl} @_{\sigma,\tau}(\lambda x : \sigma.s, t) & \succeq^{\mathcal{J}} & s[x := t] \\ \mathtt{A}_{\lambda\alpha.\sigma,\tau}(\Lambda\alpha.s) & \succeq^{\mathcal{J}} & s[\alpha := \tau] \\ \mathtt{foldl}_{\sigma}(f, s, \mathtt{nil}) & \succ^{\mathcal{J}} & s \\ \mathtt{foldl}_{\sigma}(f, s, \mathtt{cons}_{\tau}(h, t)) & \succ^{\mathcal{J}} & \mathtt{foldl}_{\sigma}(f, @_{\tau,\sigma}(@_{\sigma,\tau\to\sigma}(\mathtt{A}_{\lambda\alpha.\sigma\to\alpha\to\sigma,\tau}(f), s), h), t) \end{array}$$

Consider the first inequality; by definition it holds if $[\![@_{\sigma,\tau}(\lambda x : \sigma.s, t)]\!] \succeq [\![s[x := t]]\!]$. Since $[\![@_{\sigma,\tau}(\lambda x : \sigma.s, t)]\!] \rightsquigarrow^* [\![s]\!][x := [\![t]\!]] \oplus \mathtt{lift}_{[\![\tau]\!]}(\mathtt{flatten}_{[\![\sigma]\!]}([\![t]\!]))$, and $[\![s]\!][x := [\![t]\!]] = [\![s[x := t]]\!]$ (by Lemma 5.6), it suffices by Lemma 4.17 if $[\![s[x := t]]\!] \oplus \mathtt{lift}_{[\![\tau]\!]}(\mathtt{flatten}_{[\![\sigma]\!]}([\![t]\!])) \succeq [\![s[x := t]]\!]$. This is an instance of the general rule $u \oplus w \succeq u$ that we will obtain below.

To prove inequalities $s \succ t$ and $s \succeq t$, we will often use that $\succ$ and $\succeq$ are transitive and compatible with each other (Lem. 4.13 and 4.15), that $\rightsquigarrow \subseteq \approx$ (Lem. 4.17), that $\succeq$ is monotonic (Lem. 4.26), that both $\succ$ and $\succeq$ are monotonic over $\mathtt{lift}$ and $\mathtt{flatten}$ (Lem. 4.22 and 4.23) and that interpretations respect substitution (Lem. 5.6). We will also use Lemma 4.25 which states (among other things) that $s \succ t$ implies $s \oplus u \succ t \oplus u$. In addition, we can use the calculation rules below. The proofs may be found in [3, Appendix A.3].

▶ **Lemma 6.3.** *For all types σ and all terms s, t, u of type σ, we have:*

1. $s \oplus_{\sigma} t \approx t \oplus_{\sigma} s$ *and* $s \otimes_{\sigma} t \approx t \otimes_{\sigma} s$;
2. $s \oplus_{\sigma} (t \oplus_{\sigma} u) \approx (s \oplus_{\sigma} t) \oplus_{\sigma} u$ *and* $s \otimes_{\sigma} (t \otimes_{\sigma} u) \approx (s \otimes_{\sigma} t) \otimes_{\sigma} u$;
3. $s \otimes_{\sigma} (t \oplus_{\sigma} u) \approx (s \otimes_{\sigma} t) \oplus_{\sigma} (s \otimes_{\sigma} u)$;
4. $(\mathtt{lift}_{\sigma} 0) \oplus_{\sigma} s \approx s$ *and* $(\mathtt{lift}_{\sigma} 1) \otimes_{\sigma} s \approx s$.

▶ **Lemma 6.4.**
1. $\texttt{lift}_\sigma(n+m) \approx_\sigma (\texttt{lift}_\sigma n) \oplus_\sigma (\texttt{lift}_\sigma m)$;
2. $\texttt{lift}_\sigma(nm) \approx_\sigma (\texttt{lift}_\sigma n) \otimes_\sigma (\texttt{lift}_\sigma m)$;
3. $\texttt{flatten}_\sigma(\texttt{lift}_\sigma(n)) \approx n$.

▶ **Lemma 6.5.** *For all types σ, terms s,t of type σ and natural numbers $n > 0$:*
1. $s \oplus_\sigma t \succeq s$ *and* $s \oplus_\sigma t \succeq t$;
2. $s \oplus_\sigma (\texttt{lift}_\sigma n) \succ s$ *and* $(\texttt{lift}_\sigma n) \oplus_\sigma t \succ t$.

Note that these calculation rules immediately give the inequality $x \oplus \texttt{lift}_{nat\to\texttt{nat}}(1) \succ x$ from Example 4.20, and also that $\texttt{lift}_\sigma(n) \succ \texttt{lift}_\sigma(m)$ whenever $n > m$. By Lemmas 4.25 and 6.5 we can use *absolute positiveness*: the property that (a) $s \succeq t$ if we can write $s \approx s_1 \oplus \dots \oplus s_n$ and $t \approx t_1 \oplus \dots \oplus t_k$ with $k \le n$ and $s_i \succeq t_i$ for all $i \le k$, and (b) if moreover $s_1 \succ t_1$ then $s \succ t$. This property is typically very useful to dispense the obligations obtained in a termination proof with polynomial interpretations.

▶ **Example 6.6.** We now have the tools to finish the example of heterogeneous lists (still using the interpretation from Example 5.14). The proof obligation from Example 6.2, that $[\![@_{\sigma,\tau}(\lambda x:\sigma.s,t)]\!] \succeq [\![s[x:=t]]\!]$, is completed by Lemma 6.5(1). We have $[\![\texttt{A}_{\lambda\alpha.\sigma,\tau}(\Lambda\alpha.s)]\!] \approx [\![\Lambda\alpha.s]\!] * [\![\tau]\!] \approx [\![s[\alpha:=\tau]]\!]$ by Lemma 5.6, and $[\![\texttt{foldl}_\sigma(f,s,\texttt{nil})]\!] = [\![\texttt{nil}]\!] * [\![\sigma]\!] \cdot [\![f]\!] \cdot [\![s]\!] \oplus \texttt{lift}_{[\![\sigma]\!]}(\langle\text{something}\rangle \oplus 1) \approx [\![s]\!] \oplus \texttt{lift}_{[\![\sigma]\!]}(\langle\text{something}\rangle \oplus 1) \succ [\![s]\!]$ by Lemmas 6.4(1) and 6.5(1). For the last rule note that (using only Lemmas 4.17 and 6.4(1)):

$$
\begin{array}{l}
[\![\texttt{foldl}_\sigma(f,s,\texttt{cons}_\tau(h,t))]\!] \approx \\
[\![\texttt{cons}_\tau(h,t))]\!] * [\![\sigma]\!] \cdot [\![f]\!] \cdot [\![s]\!] \oplus \texttt{lift}_{[\![\sigma]\!]}(\texttt{flatten}([\![f]\!]) \oplus \texttt{flatten}([\![s]\!]) \oplus 1) \approx \\
(\ [\![t]\!] * [\![\sigma]\!] \cdot [\![f]\!] \cdot ([\![f]\!] * [\![\tau]\!] \cdot [\![s]\!] \cdot [\![h]\!] \oplus \texttt{lift}_{[\![\sigma]\!]}(\texttt{flatten}([\![s]\!]) \oplus \texttt{flatten}([\![h]\!]))) \oplus \\
\quad \texttt{lift}_{[\![\sigma]\!]}(\texttt{flatten}([\![f]\!] * [\![\tau]\!] \cdot [\![s]\!] \cdot [\![h]\!]) \oplus \texttt{flatten}([\![h]\!]) \oplus 1)\ ) \oplus \\
\ \texttt{lift}_{[\![\sigma]\!]}(\texttt{flatten}([\![f]\!]) \oplus \texttt{flatten}([\![s]\!]) \oplus 1) \approx \\
[\![t]\!] * [\![\sigma]\!] \cdot [\![f]\!] \cdot (\ [\![f]\!] * [\![\tau]\!] \cdot [\![s]\!] \cdot [\![h]\!] \oplus \texttt{lift}_{[\![\sigma]\!]}(\texttt{flatten}([\![s]\!]) \oplus \texttt{flatten}([\![h]\!]))\ ) \oplus \\
\ \texttt{lift}_{[\![\sigma]\!]}(\texttt{flatten}([\![f]\!] * [\![\tau]\!] \cdot [\![s]\!] \cdot [\![h]\!]) \oplus \texttt{flatten}([\![h]\!]) \oplus \texttt{flatten}([\![f]\!]) \oplus \texttt{flatten}([\![s]\!]) \oplus 2)
\end{array}
$$

On the right-hand side of the inequality, noting that $\texttt{lift}_{\sigma\to\tau}(u) \cdot w \rightsquigarrow^* \texttt{lift}_\tau(u)$, we have:

$$
\begin{array}{l}
[\![\texttt{foldl}_\sigma(f, @_{\tau,\sigma}(@_{\sigma,\tau\to\sigma}(\texttt{A}_{\lambda\alpha.\sigma\to\alpha\to\sigma,\tau}(f),s),h),t)]\!] \approx \\
\mathcal{J}(\texttt{foldl})_\sigma([\![f]\!],\ [\![f]\!] * [\![\tau]\!] \cdot [\![s]\!] \cdot [\![h]\!] \oplus \texttt{lift}_{[\![\sigma]\!]}(\texttt{flatten}([\![s]\!]) \oplus \texttt{flatten}([\![h]\!])),\ [\![t]\!]) \approx \\
[\![t]\!] * [\![\sigma]\!] \cdot [\![f]\!] \cdot (\ [\![f]\!] * [\![\tau]\!] \cdot [\![s]\!] \cdot [\![h]\!] \oplus \texttt{lift}_{[\![\sigma]\!]}(\texttt{flatten}([\![s]\!]) \oplus \texttt{flatten}([\![h]\!]))\ ) \oplus \\
\ \texttt{lift}_{[\![\sigma]\!]}(\texttt{flatten}([\![f]\!]) \oplus \texttt{flatten}([\![f]\!] * [\![\tau]\!] \cdot [\![s]\!] \cdot [\![h]\!] \oplus \\
\qquad\qquad\qquad\qquad \texttt{lift}_{[\![\sigma]\!]}(\texttt{flatten}([\![s]\!]) \oplus \texttt{flatten}([\![h]\!]))) \oplus 1) \approx \\
[\![t]\!] * [\![\sigma]\!] \cdot [\![f]\!] \cdot (\ [\![f]\!] * [\![\tau]\!] \cdot [\![s]\!] \cdot [\![h]\!] \oplus \texttt{lift}_{[\![\sigma]\!]}(\texttt{flatten}([\![s]\!]) \oplus \texttt{flatten}([\![h]\!]))\ ) \oplus \\
\ \texttt{lift}_{[\![\sigma]\!]}(\texttt{flatten}([\![f]\!]) \oplus \texttt{flatten}([\![f]\!] * [\![\tau]\!] \cdot [\![s]\!] \cdot [\![h]\!]) \oplus \texttt{flatten}([\![s]\!]) \oplus \texttt{flatten}([\![h]\!]) \oplus 1)
\end{array}
$$

Now the right-hand side is the left-hand side $\oplus$ $\texttt{lift}(1)$. Clearly, the rule is oriented with $\succ$. Thus, we may remove the last two rules, and continue the rule removal algorithm with only the first two, which together define β-reduction. This is trivial, for instance with an interpretation $\mathcal{J}(@) = \Lambda\alpha.\Lambda\beta.\lambda f.\lambda x.(f \cdot x) \oplus \texttt{lift}_\beta(\texttt{flatten}_\alpha(x) \oplus 1)$ and $\mathcal{J}(\texttt{A}) = \Lambda\alpha.\Lambda\beta.\lambda x.x * \beta \oplus \texttt{lift}_{\alpha\beta}(1)$.

7 A larger example

System $\texttt{F}$ is System $\texttt{F}_\omega$ where no higher kinds are allowed, i.e., there are no type constructors except types. By the Curry-Howard isomorphism $\texttt{F}$ corresponds to the universal-implicational fragment of intuitionistic second-order propositional logic, with the types corresponding to formulas and terms to natural deduction proofs. The remaining connectives may be encoded in $\texttt{F}$, but the permutative conversion rules do not hold [7].

In this section we show termination of the system IPC2 (see [18]) of intuitionistic second-order propositional logic with all connectives and permutative conversions, minus a few of the permutative conversion rules for the existential quantifier. The paper [18] depends on termination of IPC2, citing a proof from [27], which, however, later turned out to be incorrect. Termination of Curry-style IPC2 without $\bot$ as primitive was shown in [20]. To our knowledge, termination of the full system IPC2 remains an open problem, strictly speaking.

▶ Remark 7.1. Our method builds on the work of van de Pol and Schwichtenberg, who used higher-order polynomial interpretations to prove termination of a fragment of intuitionistic first-order logic with permutative conversions [24], in the hope of providing a more perspicuous proof of this well-known result. Notably, they did not treat disjunction, as we will do. More fundamentally, their method cannot handle impredicative polymorphism necessary for second-order logic.

The system IPC2 can be seen as a PFS with type constructors:

$$\Sigma^T_\kappa = \{\quad \bot : *, \quad \mathtt{or} : * \Rightarrow * \Rightarrow *, \quad \mathtt{and} : * \Rightarrow * \Rightarrow *, \quad \exists : (* \Rightarrow *) \Rightarrow *\}$$

We have the following function symbols:

$$\begin{array}{rcl@{\qquad}rcl}
@ & : & \forall\alpha\forall\beta.(\alpha \to \beta) \to \alpha \to \beta & \epsilon & : & \forall\alpha.\bot \to \alpha \\
\mathtt{tapp} & : & \forall\alpha : * \Rightarrow *.\forall\beta.(\forall\beta[\alpha\beta]) \to \alpha\beta & \mathtt{pr}^1 & : & \forall\alpha\forall\beta.\mathtt{and}\,\alpha\,\beta \to \alpha \\
\mathtt{pair} & : & \forall\alpha\forall\beta.\alpha \to \beta \to \mathtt{and}\,\alpha\,\beta & \mathtt{pr}^2 & : & \forall\alpha\forall\beta.\mathtt{and}\,\alpha\,\beta \to \beta \\
\mathtt{case} & : & \forall\alpha\forall\beta\forall\gamma.\mathtt{or}\,\alpha\,\beta \to (\alpha \to \gamma) \to (\beta \to \gamma) \to \gamma & \mathtt{in}^1 & : & \forall\alpha\forall\beta.\alpha \to \mathtt{or}\,\alpha\,\beta \\
\mathtt{let} & : & \forall\alpha : * \Rightarrow *.\forall\beta.(\exists(\alpha)) \to (\forall\gamma.\alpha\gamma \to \beta) \to \beta & \mathtt{in}^2 & : & \forall\alpha\forall\beta.\beta \to \mathtt{or}\,\alpha\,\beta \\
\mathtt{ext} & : & \forall\alpha : * \Rightarrow *.\forall\beta.\alpha\beta \to \exists(\alpha) & & &
\end{array}$$

The types represent formulas in intuitionistic second-order propositional logic, and the terms represent proofs. For example, a term $\mathtt{case}_{\sigma,\tau,\rho}\ s\ u\ v$ is a proof term of the formula ρ, built from a proof s of $\mathtt{or}\ \sigma\ \tau$, a proof u that σ implies ρ and a proof v that τ implies ρ. Proof terms can be simplified using 28 reduction rules, including the following (the full set of rules is available in [3, Appendix B]):

$$\begin{array}{rcl@{\quad}rcl}
@_{\sigma,\tau}(\lambda x.s, t) & \longrightarrow & s[x := t] & & & \\
\mathtt{tapp}_{\lambda\alpha.\sigma,\tau}(\Lambda\alpha.s) & \longrightarrow & s[\alpha := \tau] & \mathtt{let}_{\varphi,\rho}(\mathtt{ext}_{\varphi,\tau}(s), \Lambda\alpha.\lambda x.t) & \longrightarrow & t[\alpha := \tau][x := s] \\
\mathtt{pr}^1_{\sigma,\tau}(\mathtt{pair}_{\sigma,\tau}(s,t)) & \longrightarrow & s & \mathtt{case}_{\sigma,\tau,\rho}(\mathtt{in}^1_{\sigma,\tau}(u), \lambda x.s, \lambda y.t) & \longrightarrow & s[x := u] \\
\mathtt{pr}^2_{\sigma,\tau}(\mathtt{pair}_{\sigma,\tau}(s,t)) & \longrightarrow & t & \mathtt{case}_{\sigma,\tau,\rho}(\mathtt{in}^2_{\sigma,\tau}(u), \lambda x.s, \lambda y.t) & \longrightarrow & t[x := u]
\end{array}$$

$$\begin{array}{ll}
 & @_{\sigma,\tau}(\epsilon_{\sigma\to\tau}(s), t) \longrightarrow \epsilon_\tau(s) \\
 & \mathtt{case}_{\sigma,\tau,\rho}(\epsilon_{\mathtt{or}\,\sigma\,\tau}(u), \lambda x.s, \lambda y.t) \longrightarrow \epsilon_\rho(u) \\
 & \epsilon_\rho(\mathtt{case}_{\sigma,\tau,\bot}(u, \lambda x.s, \lambda y.t)) \longrightarrow \mathtt{case}_{\sigma,\tau,\rho}(u, \lambda x.\epsilon_\rho(s), \lambda y.\epsilon_\rho(t)) \\
 & \mathtt{pr}^2_{\rho,\pi}(\mathtt{case}_{\sigma,\tau,\mathtt{and}\,\rho,\pi}(u, \lambda x.s, \lambda y.t)) \longrightarrow \mathtt{case}_{\sigma,\tau,\pi}(u, \lambda x.\mathtt{pr}^2_{\rho,\pi}(s), \lambda y.\mathtt{pr}^2_{\rho,\pi}(t)) \\
 & \mathtt{case}_{\rho,\pi,\xi}(\mathtt{case}_{\sigma,\tau,\mathtt{or}\,\rho\,\pi}(u, \lambda x.s, \lambda y.t), \lambda z.v, \lambda a.w) \longrightarrow \\
 & \qquad \mathtt{case}_{\sigma,\tau,\xi}(u, \lambda x.\mathtt{case}_{\rho,\pi,\xi}(s, \lambda z.v, \lambda a.w), \lambda y.\mathtt{case}_{\rho,\pi,\xi}(t, \lambda z.v, \lambda a.w)) \\
 & \mathtt{let}_{\varphi,\rho}(\mathtt{case}_{\sigma,\tau,\exists\varphi}(u, \lambda x.s, \lambda y.t), v) \longrightarrow \mathtt{case}_{\sigma,\tau,\rho}(u, \lambda x.\mathtt{let}_{\varphi,\rho}(s,v), \lambda y.\mathtt{let}_{\varphi,\rho}(t,v)) \\
(*) & \mathtt{let}_{\psi,\rho}(\mathtt{let}_{\varphi,\exists\psi}(s, \Lambda\alpha.\lambda x : \varphi\alpha.t), u) \longrightarrow \mathtt{let}_{\varphi,\rho}(s, \Lambda\alpha.\lambda x : \varphi\alpha.\mathtt{let}_{\psi,\rho}(t,u))
\end{array}$$

To define an interpretation for IPC2, we will use the standard encoding of product and existential types (see [7, Chapter 11] for more details).

$$\begin{array}{rcl@{\qquad}rcl}
\sigma \times \tau & = & \forall p.(\sigma \to \tau \to p) \to p & \pi^1_{\sigma,\tau}(t) & = & t\sigma(\lambda x : \sigma.\lambda y : \tau.x) \\
\langle t_1, t_2\rangle_{\sigma,\tau} & = & \Lambda p.\lambda x : \sigma \to \tau \to p.xt_1t_2 & \pi^2_{\sigma,\tau}(t) & = & t\tau(\lambda x : \sigma.\lambda y : \tau.y) \\
\Sigma\alpha.\sigma & = & \forall p.(\forall\alpha.\sigma \to p) \to p & [\tau, t]_{\Sigma\alpha.\sigma} & = & \Lambda p.\lambda x : \forall\alpha.\sigma \to p.x\tau t \\
 & & & \mathtt{let}_\rho\, t\, \mathtt{be}\, [\alpha, x : \sigma]\, \mathtt{in}\, s & = & t\rho(\Lambda\alpha.\lambda x : \sigma.s)
\end{array}$$

We do not currently have an algorithmic method to find a suitable interpretation. Instead, we used the following manual process. We start by noting the minimal requirements given by the first set of rules (e.g., that $\mathtt{pr}^1_{\sigma,\tau}(\mathtt{pair}_{\sigma,\tau}(s,t)) \succeq s$); to orient these inequalities, it would be good to for instance have $[\![\mathtt{pair}_{\sigma,\tau}(s,t)]\!] \succeq \langle [\![s]\!], [\![t]\!]\rangle_{[\![\sigma]\!],[\![\tau]\!]}$ and $[\![\mathtt{pr}^i_{\sigma,\tau}(s)]\!] = \pi^i_{[\![\sigma]\!],[\![\tau]\!]}([\![s]\!])$. To make the interpretation safe, we additionally include clauses $\mathtt{lift}(\mathtt{flatten}(x))$ for any unsafe arguments x; to make the rules *strictly* oriented, we include clauses $\mathtt{lift}(1)$. Unfortunately, this approach does not suffice to orient the rules where some terms are duplicated, such as the second- and third-last rules. To handle these rules, we *multiply* the first argument of several symbols with the second (and possibly third). Some further tweaking gives the following safe interpretation, which orients most of the rules:

$$\begin{array}{rclrcl} \mathcal{TM}(\bot) & = & \mathtt{nat} & \mathcal{TM}(\mathtt{and}) & = & \lambda\alpha_1\lambda\alpha_2.\alpha_1 \times \alpha_2 \\ \mathcal{TM}(\exists) & = & \lambda(\alpha : * \Rightarrow *).\Sigma\gamma.\alpha\gamma & \mathcal{TM}(\mathtt{or}) & = & \lambda\alpha_1\lambda\alpha_2.\alpha_1 \times \alpha_2 \end{array}$$

$$\begin{array}{rcll} \mathcal{J}(\epsilon) & = & \Lambda\alpha : *.\lambda x : \mathtt{nat}. & \mathtt{lift}_\alpha(2 \otimes x \oplus 1) \\ \mathcal{J}(@) & = & \Lambda\alpha\Lambda\beta\lambda x : \alpha \to \beta.\lambda y : \alpha. & \mathtt{lift}_\beta(2) \otimes (x \cdot y) \oplus \mathtt{lift}_\beta(\mathtt{flatten}_\alpha(y) \oplus \\ & & & \quad \mathtt{flatten}_{\alpha\to\beta}(x) \otimes \mathtt{flatten}_\beta(y) \oplus 1) \\ \mathcal{J}(\mathtt{tapp}) & = & \Lambda\alpha : * \Rightarrow *.\Lambda\beta.\lambda x : \forall\gamma.\alpha\gamma. & \mathtt{lift}_{\alpha\beta}(2) \otimes (x * \beta) \oplus \mathtt{lift}_{\alpha\beta}(1) \\ \mathcal{J}(\mathtt{ext}) & = & \Lambda\alpha : * \Rightarrow *.\Lambda\beta : *.\lambda x : \alpha\beta. & [\beta, x] \oplus \mathtt{lift}_{\Sigma\gamma.\beta\gamma}(\mathtt{flatten}_{\alpha\gamma}(x)) \\ \mathcal{J}(\mathtt{pair}) & = & \Lambda\alpha\Lambda\beta\lambda x : \alpha, y : \beta. & \langle x, y\rangle \oplus \mathtt{lift}_{\alpha\times\beta}(\mathtt{flatten}_\alpha(x) \oplus \mathtt{flatten}_\beta(y)) \\ \mathcal{J}(\mathtt{pr}^1) & = & \Lambda\alpha\Lambda\beta\lambda x : \alpha \times \beta. & \mathtt{lift}_\alpha(2) \otimes \pi^1(x) \oplus \mathtt{lift}_\alpha(1) \\ \mathcal{J}(\mathtt{pr}^2) & = & \Lambda\alpha\Lambda\beta\lambda x : \alpha \times \beta. & \mathtt{lift}_\beta(2) \otimes \pi^2(x) \oplus \mathtt{lift}_\beta(1) \\ \mathcal{J}(\mathtt{in}^1) & = & \Lambda\alpha\Lambda\beta\lambda x : \alpha. & \langle x, \mathtt{lift}_\beta(1)\rangle \oplus \mathtt{lift}_{\alpha\times\beta}(\mathtt{flatten}_\alpha(x)) \\ \mathcal{J}(\mathtt{in}^2) & = & \Lambda\alpha\Lambda\beta\lambda x : \beta. & \langle \mathtt{lift}_\alpha(1), x\rangle \oplus \mathtt{lift}_{\alpha\times\beta}(\mathtt{flatten}_\beta(x)) \end{array}$$

$$\begin{array}{rcl} \mathcal{J}(\mathtt{let}) & = & \Lambda\alpha : * \Rightarrow *.\Lambda\beta : *.\lambda x : \Sigma\xi.\alpha\xi, y : \forall\xi.\alpha\xi \to \beta. \\ & & \mathtt{lift}_\beta(1) \oplus \mathtt{lift}_\beta(2) \otimes (\mathtt{let}_\beta\, x\, \mathtt{be}\, [\xi, z]\, \mathtt{in}\, y\xi z) \oplus \\ & & \mathtt{lift}_\beta(\mathtt{flatten}_{\Sigma\gamma.\alpha\gamma}(x) \oplus 1) \otimes (y * \mathtt{nat} \cdot \mathtt{lift}_{\alpha\mathtt{nat}}(0)) \\ \mathcal{J}(\mathtt{case}) & = & \Lambda\alpha, \beta, \xi.\lambda x : \alpha \times \beta, y : (\alpha \to \xi), z : (\beta \to \xi). \\ & & \quad \mathtt{lift}_\xi(2) \oplus \mathtt{lift}_\xi(3 \otimes \mathtt{flatten}_{\alpha\times\beta}(x)) \oplus \\ & & \qquad \mathtt{lift}_\xi(\mathtt{flatten}_{\alpha\times\beta}(x) \oplus 1) \otimes (y \cdot \pi^1(x) \oplus z \cdot \pi^2(x)) \end{array}$$

Above, $\otimes$ binds stronger than $\oplus$. The derivations to orient rules with these interpretations are also given in [3, Appendix B].

The only rules that are not oriented with this interpretation – not with $\succeq$ either – are the ones of the form $f(\mathtt{let}(s,t),\dots) \longrightarrow \mathtt{let}(s, f(t,\dots))$, like the rule marked (*) above. Nonetheless, this is already a significant step towards a systematic, extensible methodology of termination proofs for IPC2 and similar systems of higher-order logic. Verifying the orientations is still tedious, but our method raises hope for at least partial automation, as was done with polynomial interpretations for non-polymorphic higher-order rewriting [6].

8 Conclusions and future work

We introduced a powerful and systematic methodology to prove termination of higher-order rewriting with full impredicative polymorphism. To use the method one just needs to invent safe interpretations and verify the orientation of the rules with the calculation rules.

As the method is tedious to apply manually for larger systems, a natural direction for future work is to look into automation: both for automatic verification that a given interpretation suffices and – building on existing termination provers for first- and higher-order term rewriting – for automatically finding a suitable interpretation.

In addition, it would be worth exploring improvements of the method that would allow us to handle the remaining rules of IPC2, or extending other techniques for higher-order termination such as orderings (see, e.g., [11]) or dependency pairs (e.g., [13, 19]).

References

1. F. Blanqui. Definitions by rewriting in the Calculus of Constructions. *MSCS*, 15(1):37–92, 2005.
2. D. Cousineau and G. Dowek. Embedding Pure Type Systemsin the lambda-Pi-calculus modulo. In *TLCA*, pages 102–117, 2017.
3. Ł. Czajka and C. Kop. Polymorphic Higher-Order Termination (extended version), 2019. arXiv:1904.09859.
4. G. Dowek. Models and termination of proof reduction in the λΠ-calculus modulo theory. In *ICALP*, pages 109:1–109:14, 2017.
5. M. Fiore and M. Hamana. Multiversal Polymorphic Algebraic Theories: syntacs, semantics, translations and equational logic. In *LICS*, pages 520–520, 2013.
6. C. Fuhs and C. Kop. Polynomial Interpretations for Higher-Order Rewriting. In *RTA*, pages 176–192, 2012.
7. J.-V. Girard, P. Taylor, and Y. Lafont. *Proofs and Types.* Cambridge University Press, 1989.
8. J.-Y. Girard. Une Extension De l'Interpretation De Gödel a l'Analyse, Et Son Application a l'Elimination Des Coupures Dans l'Analyse Et La Theorie Des Types. In *SLS*, pages 63–92. Elsevier, 1971.
9. M. Hamana. Polymorphic Rewrite Rules: Confluence, Type Inference, and Instance Validation. In *FLOPS*, pages 99–115, 2018.
10. B. Jacobs and J. Rutten. An introduction to (co)algebras and (co)induction. In *Advanced Topics in Bisimulation and Coinduction*, pages 38–99. Cambridge University Press, 2011.
11. J. Jouannaud and A. Rubio. Polymorphic higher-order recursive path orderings. *JACM*, 54(1):1–48, 2007.
12. C. Kop. *Higher Order Termination.* PhD thesis, VU University Amsterdam, 2012.
13. C. Kop and F. van Raamsdonk. Dynamic Dependency Pairs for Algebraic Functional Systems. *LMCS*, 8(2):10:1–10:51, 2012.
14. D. Kozen and A. Silva. Practical coinduction. *Mathematical Structures in Computer Science*, 27(7):1132–1152, 2017.
15. J.C. van de Pol. *Termination of Higher-order Rewrite Systems.* PhD thesis, University of Utrecht, 1996.
16. D. Sangiorgi. *Introduction to Bisimulation and Coinduction.* Cambridge University Press, 2012.
17. M.H. Sørensen and P. Urzyczyn. *Lectures on the Curry-Howard Isomorphism.* Elsevier, 2006.
18. M.H. Sørensen and P. Urzyczyn. A Syntactic Embedding of Predicate Logic into Second-Order Propositional Logic. *Notre Dame Journal of Formal Logic*, 51(4):457–473, 2010.
19. S. Suzuki, K. Kusakari, and F. Blanqui. Argument Filterings and Usable Rules in Higher-Order Rewrite Systems. *IPSJ Transactions on Programming*, 4(2):1–12, 2011.
20. M. Tatsuta. Simple Saturated Sets for Disjunction and Second-Order Existential Quantification. In *TLCA 2007*, pages 366–380, 2007.
21. Terese. *Term Rewriting Systems.* Cambridge University Press, 2003.
22. A.S. Troelstra and H. Schwichtenberg. *Basic Proof Theory.* Cambridge University Press, 1996.
23. J.C. van de Pol. Termination Proofs for Higher-order Rewrite Systems. In *HOA*, pages 305–325, 1993.
24. J.C. van de Pol and H. Schwichtenberg. Strict Functionals for Termination Proofs. In *TLCA 95*, pages 350–364, 1995.
25. D. Wahlstedt. *Type Theory with First-Order Data Types and Size-Change Termination.* PhD thesis, Göteborg University, 2004.
26. D. Walukiewicz-Chrząszcz. Termination of rewriting in the Calculus of Constructions. *JFP*, 13(2):339–414, 2003.
27. A. Wojdyga. Short Proofs of Strong Normalization. In *MFCS*, pages 613–623, 2008.

On the Taylor Expansion of Probabilistic λ-terms

Ugo Dal Lago
University of Bologna, Italy
INRIA Sophia Antipolis, France
ugo.dallago@unibo.it

Thomas Leventis
INRIA Sophia Antipolis, France
thomas.leventis@ens-lyon.org

Abstract

We generalise Ehrhard and Regnier's Taylor expansion from *pure* to *probabilistic* λ-terms. We prove that the Taylor expansion is adequate when seen as a way to give semantics to probabilistic λ-terms, and that there is a precise correspondence with probabilistic Böhm trees, as introduced by the second author. We prove this adequacy through notions of probabilistic resource terms and explicit Taylor expansion.

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

Keywords and phrases Probabilistic Lambda-Calculi, Taylor Expansion, Linear Logic

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.13

Related Version A full version of the paper is available at `http://arxiv.org/abs/1904.09650`.

Funding The authors are partially supported by the ERC Consolidator Grant DLV-818616 DIAPASoN, as well as by the ANR projects 14CE250005 ELICA and 16CE250011 REPAS.

1 Introduction

Linear logic is a proof-theoretical framework which, since its inception [10], has been built around an analogy between on the one hand linearity in the sense of linear algebra, and on the other hand the absence of copying and erasing in cut elimination and higher-order rewriting. This analogy has been pushed forward by Ehrhard and Regnier, who introduced a series of logical and computational frameworks accounting, along the same analogy, for concepts like that of a differential, or the very related one of an approximation. We are implicitly referring to differential λ-calculus [6], to differential linear logic [8], and to the Taylor expansion of ordinary λ-terms [9]. The latter has given rise to an extremely interesting research line, with many deep contributions in the last ten years. Not only the Taylor expansion of pure λ-terms has been shown to be endowed with a well-behaved notion of reduction, but the Böhm tree and Taylor expansion operators are now known to commute [7]. This easily implies that the equational theory (on pure λ-terms) induced by the Taylor expansion coincides with the one induced by Böhm trees.

The Taylor expansion operator is essentially *quantitative*, in that its codomain is not merely the set of resource terms [3, 6], a term syntax for promotion-free differential proofs, but the set of *linear combinations* of those terms, with positive real number coefficients. When enlarging the domain of the operator to account for a more quantitative language, one is naturally lead to consider algebraic λ-calculi, to which giving a clean computational meaning has been proved hard so far [18].

But what about *probabilistic* λ-calculi [11], which have received quite some attention recently (see, e.g. [5, 2, 16]) due to their applicability to randomised computation and bayesian programming? Can the Taylor expansion naturally be generalised to those calculi? This

© Ugo Dal Lago and Thomas Leventis;
licensed under Creative Commons License CC-BY
4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019).
Editor: Herman Geuvers; Article No. 13; pp. 13:1–13:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

is an interesting question, to which we give the first definite positive answer in this paper. In particular, we show that the Taylor expansion of probabilistic λ-terms is a conservative extension of the well-known one on ordinary λ-terms. In particular, the target can be taken, as usual, as a linear combination of *ordinary* resource terms, i.e., the same kind of structure which Ehrhard and Regnier considered in their work on the Taylor expansion of *pure* λ-terms. We moreover show that the Taylor expansion, as extended to probabilistic λ-terms, continues to enjoy the nice properties it has in the deterministic realm. In particular, it is adequate as a way to give semantics to probabilistic λ-terms, and the equational theory on probabilistic λ-terms induced by Taylor expansion coincides with the one induced by a probabilistic variation on Böhm trees [1]. The latter, noticeably, has been proved to capture observational equivalence, one quotiented modulo η-equivalence [1].

Are we the first ones to embark on the challenge of generalising Taylor's expansion to probabilistic λ-calculi, and in general to effectful calculi? Actually, some steps in this direction have recently been taken. First of all, we need to mention the line of works originated by Tsukada and Ong's paper on rigid resource terms [14]. This has been claimed from the very beginning to be a way to model effects in the resource calculus, but it has also been applied to, among others, probabilistic effects, giving rise to quantitative denotational models [15]. The obtained models are based on species, and are proved to be adequate. The construction being generic, there is no aim at providing a precise comparison between the discriminating power of the obtained theory and, say, observational equivalence: the choice of the underlying effect can in principle have a huge impact on it.

One should also mention Vaux's work on the algebraic λ-calculus [18], where one can build arbitrary linear combinations of terms. He showed a correspondence between Taylor expansion and Böhm trees, but only for terms whose Böhm trees approximants at finite depths are computable in a finite number of steps. This includes all ordinary λ-terms but not all probabilistic ones. More recently Olimpieri and Vaux have studied a Taylor expansion for a non-deterministic λ-calculus [19] corresponding to our notion of *explicit* Taylor expansion (Section 3).

The Probabilistic Taylor Expansion, Informally

The main idea behind building the Taylor expansion of any λ-term M is to describe the dynamics of M by way of *linear approximations* of M. In the realm of the λ-calculus, a linear approximation has traditionally been taken as a *resource term*:

$$s, t \in \Delta := x \mid \lambda x.s \mid \langle s\rangle\ \bar{t} \qquad\qquad \bar{s}, \bar{t} \in \Delta^! := [s_1, \ldots, s_n].$$

A resource term can be seen as a pure λ-term in which applications have the form $\langle s\rangle\ \bar{t}$, where s is a term and $\bar{t}$ is a *multiset* of terms, and in which the result of firing the redex $\langle \lambda x.s\rangle\ \bar{t}$ is the linear combination of all the terms obtained by allocating the resources in $\bar{t}$ to the occurrences of x in s. For instance, one such element in the Taylor expansion of δ is $\lambda x.(\langle x\rangle\ [x])$, where the occurrence of x in head position is provided with only one copy of its argument. If applied to the multiset $[y, z]$, this term would reduce into $\langle y\rangle\ [z] + \langle z\rangle\ [y]$. Similarly, an element in the Taylor expansion of $\delta\ I$ would be $\langle \lambda x.\langle x\rangle\ [x]\rangle\ [I^2]$, which reduces into $2.\langle I\rangle\ [I]$. Another element of the same Taylor expansion is $\langle \lambda x.\langle x\rangle\ [x]\rangle\ [I^3]$, but this one reduces into 0: there is no way to use its resources linearly, i.e., using them without copying and erasing.

The actual Taylor expansion of a term is built by translating any application $M\ N$ into an infinite sum

$$(M\ N)^* = \sum_{n\in\mathbb{N}} \frac{1}{n!}.\langle M^*\rangle\ [(N^*)^n]$$

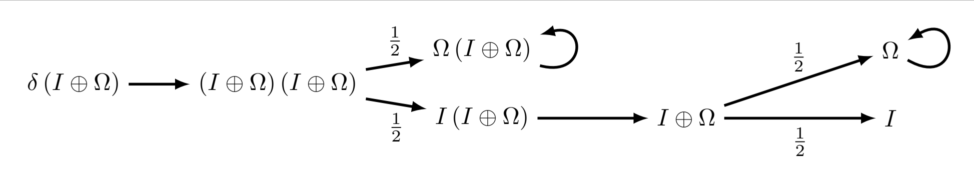


Figure 1 M's Reduction Tree.

and $x^* = x$, $(\lambda x.M)^* = \lambda x.M^*$. Remark that M^* and N^* are linear combinations, but constructors of the resource calculus are multilinear. For instance $(\lambda x.M)^* = \sum_{s\in\Delta} M^*_s.\lambda x.s$. As an example the Taylor expansion of $\delta\ I$ is $\sum_{m,n\in\mathbb{N}} \frac{1}{m!n!}.\langle\lambda x.\langle x\rangle\ [x^m]\rangle\ [I^n]$. Remark that any summand properly reduces only when $n = m+1$, in which case it reduces to $n!.\langle I\rangle\ [I^m]$. In turn $\langle I\rangle\ [I^m]$ reduces properly only when $m = 1$, and the result is I. All the other terms reduce to 0. In the end the Taylor expansion of $\delta\ I$ normalises to $\frac{2!}{1!2!}.I = I$, which is exactly the Taylor expansion of the normal form of $\delta\ I$. More generally every Taylor expansion is normalisable, and the normal form of the Taylor expansion of a term corresponds to the Taylor expansion of its Böhm tree, hence (normal forms of) Taylor expansions yields an interesting model of deterministic λ-calculus.

Now let us consider the probabilistic λ-term $M = \delta(I \oplus \Omega)$, where $\oplus$ is an operator for binary, fair, probabilistic choice, $\delta = \lambda x.xx$, $I = \lambda.x.x$ and $\Omega = \delta\delta$ is a purely diverging, term. As such, M is a term of a minimal, untyped, probabilistic λ-calculus. Evaluation of M is performed leftmost-outermost is as in Figure 1. In particular, the probability of convergence for M is $\frac{1}{4}$. Please observe that two copies of the argument $I \oplus \Omega$ are produced, and that the "rightmost" one is evaluated only when the "leftmost" one converges, i.e. when the probabilistic choice $I \oplus \Omega$ produces I as a result.

Extending the Taylor expansion to probabilistic terms seems straightforward, a natural candidate for the Taylor expansion of $M \oplus N$ being just $\frac{1}{2}.M^* + \frac{1}{2}.N^*$. When computing the Taylor expansion of M we will find expressions such as $\langle\lambda x.\langle x\rangle\ [x]\rangle\ [(\frac{1}{2}.I + \frac{1}{2}.\Omega^*)^2]$, i.e. $\frac{1}{4}.\langle\lambda x.\langle x\rangle\ [x]\rangle\ [I^2] + \frac{1}{4}.\langle\lambda x.\langle x\rangle\ [x]\rangle\ [\Omega^2] + \frac{1}{2}.\langle\lambda x.\langle x\rangle\ [x]\rangle\ [I, \Omega]$. For non-trivial reasons, the Taylor expansion of any diverging term normalises to 0, so just like in our previous example, the only element in M^* which does not reduce to 0 is $\langle\lambda x.\langle x\rangle\ [x]\rangle\ [I^2]$. The difference is that this time it appears with a coefficient $\frac{1}{1!2!}\frac{1}{4}$, so M^* normalises to $\frac{1}{4}.I$. Please notice how this is once again the "normal form" of the original term M.

The goal of this paper is to show that the correspondence between Taylor expansions and Böhm trees is preserved when we add probabilistic choices to the calculus. Unfortunately although the final result is the same as in the deterministic setting, the known proof techniques fail. To begin with not all (infinite) linear combinations of resource term are normalisable. Normal forms of deterministic Taylor expansions always exist because such expansions are *uniform*: all terms in their support have the same shape (the support of $(\lambda x.M)^*$ only contains abstractions, etc.) and this property is known to ensure normalisability. This property does not hold for *probabilistic* expansions: $(x \oplus \lambda y.y)^*$ contains both the variable x and the abstraction $\lambda y.y$. Thus we need to find a different way to prove that probabilistic Taylor expansions normalise. We proceed by using an intermediate notion of Taylor expansion *with explicit choices*, which enjoys uniformity and share other properties with deterministic Taylor expansion, and we then transpose directly the results on this intermediate construction to the "natural" probabilistic Taylor expansion.

Layout of the Paper

In Section 2 we introduce the resource calculus with explicit choices and show it enjoys the same properties as the usual deterministic resource calculus. In Section 3, we define the explicit Taylor expansion from probabilistic λ-terms. These constructions have an interest in themselves and they have been independently studied by Olimpieri and Vaux [19] for a non-deterministic calculus but in this paper they are just an intermediate step towards proving our main results. Definitionally, the crux of the paper is Section 4, in which the Taylor expansion of a probabilistic λ-term is made to produce *ordinary* resource terms. The relation between Böhm trees and Taylor expansions is investigated in Section 5 and Section 6.

Notations

We write $\mathbb{N}$ for the set of natural numbers and $\mathbb{R}^+$ for the set of nonnegative real numbers. Given a set A, we write $\mathbb{R}^+\langle A\rangle$ for the set of families of positive real numbers indexed by elements in A. We write such families as linear combinations: an element $S \in \mathbb{R}^+\langle A\rangle$ is a sum $S = \sum_{a\in A} S_a.a$, with $S_a \in \mathbb{R}^+$. The support of a family $S \in \mathbb{R}^+\langle A\rangle$ is $\mathrm{supp}(S) = \{a \in A \mid S_a > 0\}$. We write $\mathbb{R}^+[A]$ for those families $S \in \mathbb{R}^+\langle A\rangle$ such that $\mathrm{supp}(S)$ is finite. Given $a \in A$ we often write a for $1.a \in \mathbb{R}^+\langle A\rangle$ unless we want to emphasise the difference between the two expressions. We also define *finite multisets* over A as functions $m : A \to \mathbb{N}$ such that $m(a) \neq 0$ for finitely many $a \in A$. We use the notation $[a_1, \dots, a_n]$ to describe the multiset m such that $m(a)$ is the number of indices $i \leq n$ such that $a_i = a$.

2 Probabilistic Resource Calculus

In this section, we describe the theory of resource terms with explicit choices, for the purpose of extending many of the properties of resource terms to the probabilistic case. All this has an interest in itself, but here this is mainly useful as a way to render certain proofs about the Taylor Expansion easier (see Section 3 for more details). For this reason we try to give the reader a clear understanding of this calculus and of why these definitions and properties are useful, without focusing on the actual proofs. These are straightforward generalisations of those for deterministic resource terms [9] and can be found in an extended version of this paper [4]. The same results have recently been given for a non-deterministic calculus [19] by Olimpieri and Vaux.

2.1 The Basics

▶ **Definition 1.** *The sets of* ***probabilistic simple resource terms*** $\Delta_\oplus$ *and of* ***probabilistic simple resource poly-terms*** $\Delta_\oplus^!$ *over a set of variables* $\mathcal{V}$ *are defined by mutual induction as follows:*

$$s, t \in \Delta_\oplus := x \mid \lambda x.s \mid \langle s\rangle\ \bar{t} \mid s \oplus_p \bullet \mid \bullet \oplus_p s \qquad \bar{s}, \bar{t} \in \Delta_\oplus^! := [s_1, \dots, s_n]$$

where p ranges over $[0,1]$. We call ***finite probabilistic resource terms*** *the finite linear combinations of resource terms in $\mathbb{R}^+[\Delta_\oplus]$, and* ***finite probabilistic resource poly-terms*** *the finite linear combinations of resource poly-terms in $\mathbb{R}^+[\Delta_\oplus^!]$. We extend the constructors of simple (poly-)terms to (poly-)terms by linearity, e.g., if $S \in \mathbb{R}^+[\Delta_\oplus]$ then $\lambda x.S$ is defined as the poly-term such that $(\lambda x.S)_{\lambda x.s} = S_s$ and $(\lambda x.S)_t = 0$ if t is not an abstraction.*

Some consecutive abstractions $\lambda x_1 \dots . \lambda x_n.s$ will be indicated as $\lambda x_1 \dots x_n.s$, or even as $\lambda\vec{x}.s$. Similarly, to describe many successive applications $\langle\langle\langle M\rangle\ N_1\rangle\ \dots\rangle\ N_k$, we use a single pair of brackets and we write $\langle M\rangle\ N_1\ \dots\ N_k$.

We write $\Delta_\oplus^{(!)}$ for $\Delta_\oplus \cup \Delta_\oplus^!$, which is ranged over by metavariables like σ, τ. Note that intuitively $\Delta_\oplus^{(!)}$ should stand for either $\Delta_\oplus$ or $\Delta_\oplus^!$, not their union. For instance we will prove some properties for finite linear combinations in $\mathbb{R}^+[\Delta_\oplus^{(!)}]$, but the only relevant linear combinations are the actual (poly-)terms in $\mathbb{R}^+[\Delta_\oplus]$ or $\mathbb{R}^+[\Delta_\oplus^!]$. Yet this distinction is technically irrelevant, and all our results hold if we define $\Delta_\oplus^{(!)}$ as a union.

The reason why linear combinations over such elements are dubbed *terms* will be clear once we describe the operational semantics of the resource calculus. The main point of the resource calculus is to allow functions to use their argument arbitrarily many times and yet remain entirely linear, which is achieved by taking multisets as arguments: if a function uses its argument n times then it needs to receive n resources as argument and use each of them linearly. This idea has two consequences. First, an application can fail if a function is not given exactly as many arguments as it needs, as it would need either to duplicate or to discard some of them. Second, the result of a valid application is often not unique: a function can choose how to allocate the different resources to the different calls to its argument, and different choices may lead to different results. Both these features are treated using linear combinations: a failed application results in 0 (i.e. the trivial linear combination) and a successful one yields the sum of all its possible outcomes.

▶ **Definition 2.** *We define the substitution of $\bar{t} \in \Delta_\oplus^!$ for $x \in \mathcal{V}$ in $s \in \Delta_\oplus$ by:*

$$\delta_x s \cdot [t_1, \ldots, t_n] = \begin{cases} 0 \text{ if } s \text{ does not have exactly } n \text{ free occurences of } x \\ \sum_{\rho \in \mathfrak{S}_n} s[t_{\rho(1)}/x_1, \ldots, t_{\rho(n)}/x_n] \in \mathbb{R}^+[\Delta_\oplus^{(!)}] \text{ otherwise} \end{cases}$$

where $x_1, \ldots, x_n$ are the free occurrences of x in s and $\mathfrak{S}_n$ is the set of permutations over $\{1, \ldots, n\}$.

▶ **Example 3.** A basic example is $\delta_x(\langle x \rangle\ [x]) \cdot [y, z] = \langle y \rangle\ [z] + \langle z \rangle\ [y]$: there are two occurrences of x in $\langle x \rangle\ [x]$, so there are two ways to substitute $[y, z]$ for them. Remark that we also have $\delta_x[x, x] \cdot [y, z] = [y, z] + [z, y] = 2.[y, z]$: the two occurrences of x are not as clearly distinguished as in the first example but they still count as different occurrences. Similarly $\delta_x(\langle x \rangle\ [x]) \cdot [y, y] = 2.\langle y \rangle\ [y]$ and $\delta_x[x, x] \cdot [y, y] = 2.[y, y]$: there are two distinct occurrences of y, so there are two ways to allocate them. As another example, please consider $\delta_x(\lambda x.x) \cdot [y] = \delta_x(\langle x \rangle\ [x]) \cdot [y] = 0$: the substitution fails if the number of resources does not match the number of free occurrences of the substituted variable.

The operational semantics of the deterministic resource calculus [9] is usually given as a single rule of β-reduction. In the probabilistic setting, we also need rules to make choices commute with head contexts.

▶ **Definition 4.** *The reductions $\to_\beta$ and $\to_\oplus$ are defined from $\Delta_\oplus^{(!)}$ to $\mathbb{R}^+[\Delta_\oplus^{(!)}]$ by:*

$$\langle \lambda x.s \rangle\ \bar{t} \to_\beta \delta_x s \cdot \bar{t}$$

$$\lambda x.(s \oplus_p \bullet) \to_\oplus \lambda x.s \oplus_p \bullet \qquad \lambda x.(\bullet \oplus_p s) \to_\oplus \bullet \oplus_p \lambda x.s$$

$$\langle s \oplus_p \bullet \rangle\ \bar{t} \to_\oplus \langle s \rangle\ \bar{t} \oplus_p \bullet \qquad \langle \bullet \oplus_p s \rangle\ \bar{t} \to_\oplus \bullet \oplus_p \langle s \rangle\ \bar{t}$$

extended under arbitrary contexts. We simply write $\to$ for $\to_\beta \cup \to_\oplus$. Reduction can be extended to finite terms in the following way: if $S \in \mathbb{R}^+[\Delta_\oplus^{(!)}]$, $S_\sigma > 0$ and $\sigma \to T$ then $S \to S - S_\sigma.\sigma + S_\sigma T$.

As the resource calculus does not allow any duplication, and β-reduction erases some constructors, it naturally decreases the size of the involved simple terms. Consequently, β-reduction is strongly normalising. This result can be extended to the whole reduction $\to$, which is also confluent.

▶ **Proposition 5.** *The reduction $\to$ is confluent and strongly normalising on $\mathbb{R}^+[\Delta_\oplus^{(!)}]$. Given $S \in \mathbb{R}^+[\Delta_\oplus^{(!)}]$ we write $\mathrm{nf}(S)$ for its unique normal form for $\to$, and given $\sigma \in \Delta_\oplus^{(!)}$ we write $\mathrm{nf}(\sigma)$ for $\mathrm{nf}(1.\sigma)$.*

2.2 Infinite Terms

So far we only worked with finite terms but to fully express the operational behaviour of a λ-term in the resource λ-calculus, which is the purpose of the Taylor expansion, we need infinite ones. We can extend the constructors of the calculus to $\mathbb{R}^+\langle\Delta_\oplus^{(!)}\rangle$ by linearity and generalise the reduction relation $\to$, but Proposition 5 fails. Indeed let $I_0 = I = \lambda x.x$ and $I_{n+1} = \langle I_n\rangle\ [I]$. For $n \in \mathbb{N}$, let $S = \sum_{n\in\mathbb{N}} I_n$. Then, for all $n \in \mathbb{N}$ the term I_n normalises in n steps and S *does not* normalise in a finite number of reduction steps. A simple solution to this problem is to define the "normal form" of an infinite term by normalising each of its components: we can set $\mathrm{nf}(S) = \sum_{\sigma\in\Delta_\oplus^{(!)}} S_\sigma \mathrm{nf}(\sigma)$. But then another problem arises. In our previous example, we have $\mathrm{nf}(I_n) = I$ for all $n \in \mathbb{N}$, thus we would have $\mathrm{nf}(S) = \sum_{n\in\mathbb{N}} I$, which is not an element of $\mathbb{R}^+\langle\Delta_\oplus^{(!)}\rangle$ as the coefficient of I is infinite. Still we can use this pointwise normalisation if we consider terms with a particular property, called *uniformity*.

▶ **Definition 6.** *The **coherence relation** $\coh$ on $\Delta_\oplus^{(!)}$ is defined by:*

$$\frac{}{x \coh x} \qquad \frac{s \coh s'}{\lambda x.s \coh \lambda x.s'} \qquad \frac{s \coh s' \quad \overline{t} \coh \overline{t'}}{\langle s\rangle\ \overline{t} \coh \langle s'\rangle\ \overline{t'}} \qquad \frac{s \coh s'}{s \oplus_p \bullet \coh s' \oplus_p \bullet} \qquad \frac{s \coh s'}{\bullet \oplus_p s \coh \bullet \oplus_p s'}$$

$$\frac{s \coh s \quad t \coh t}{s \oplus_p \bullet \coh \bullet \oplus_p t} \qquad \frac{s \coh s \quad t \coh t}{\bullet \oplus_p t \coh s \oplus_p \bullet} \qquad \frac{\forall i, j \leq m+n, s_i \coh s_j}{[s_1, \ldots, s_m] \coh [s_{m+1}, \ldots, s_{m+n}]}$$

*For $S, S' \in \Delta_\oplus^{(!)}$ we write $S \coh S'$ when for all $\sigma, \sigma' \in \mathrm{supp}(S) \cup \mathrm{supp}(S')$, $\sigma \coh \sigma'$. A simple (poly-)term $\sigma \in \Delta_\oplus^{(!)}$ is called **uniform** if $\sigma \coh \sigma$, and a term $S \in \mathbb{R}^+\langle\Delta_\oplus^{(!)}\rangle$ is called **uniform** if $S \coh S$.*

Observe that the term S defined above is not uniform: I_0 is an abstraction whereas for any $n \in \mathbb{N}$, I_{n+1} is an application, hence I_0 and I_{n+1} are not coherent. More generally I_m and I_n are not coherent whenever $m \neq n$. We can change the definition of I_0 to get a uniform term: let $I'_0 = \langle I\rangle\ [\,]$, $I'_{n+1} = \langle I'_n\rangle\ [I]$ for $n \in \mathbb{N}$, and $S' = \sum_{n\in\mathbb{N}} I'_n$. Then I'_{n+1} reduces into I'_n, just like I_{n+1} reduces into I_n, but I'_0 is not a normal form as it reduces into 0. Thus the uniform term S' has a normal form $\mathrm{nf}(S') = \sum_{n\in\mathbb{N}} \mathrm{nf}(I'_n) = 0$.

▶ Remark 7. In the rules for $s \oplus_p \bullet \coh \bullet \oplus_p t$ and $\bullet \oplus_p t \coh s \oplus_p \bullet$ we require $s \coh s$ and $t \coh t$ to ensure that whenever $\sigma \coh \tau$, the simple (poly-)terms σ and τ are necessarily uniform. This is not crucial as we usually consider uniform (poly-)terms (whose support only contains uniform simple (poly-)terms), and indeed in [19] the non-deterministic terms $s \oplus \bullet$ and $\bullet \oplus t$ are always considered coherent. We only add this requirements to simplify inductive reasoning on $\coh$.

What makes coherence and uniformity interesting is that if two coherent terms S and S' have disjoint supports, then all of their reducts, and in particular their normal forms, have disjoint supports. Then any element in the support of $\mathrm{nf}(S + S')$ comes either from $\mathrm{nf}(S)$ or from $\mathrm{nf}(S')$, but it cannot come from both.

▶ **Proposition 8.** *Given $S, S' \in \mathbb{R}^+[\Delta_\oplus^{(!)}]$, if $S \coh S'$ then $\mathrm{nf}(S) \coh \mathrm{nf}(S')$. If moreover $\mathrm{supp}(S) \cap \mathrm{supp}(S') = \emptyset$ then $\mathrm{supp}(\mathrm{nf}(S)) \cap \mathrm{supp}(\mathrm{nf}(S')) = \emptyset$.*

This immediately implies that pointwise reduction of infinite uniform terms is well defined, as both complete left reducts and normal forms of distinct but coherent simple (poly-)terms have disjoint supports.

▶ **Corollary 9.** *If $S \in \mathbb{R}^+\langle\Delta_\oplus^{(!)}\rangle$ is uniform then $\sum_{\sigma\in\Delta_\oplus^{(!)}} S_\sigma \mathrm{nf}(\sigma)$ is in $\mathbb{R}^+\langle\Delta_\oplus^{(!)}\rangle$. We write $\mathrm{nf}(S)$ for this sum.*

2.3 Regular Terms

The deterministic Taylor expansion associates to any λ-term a *uniform* term, and explicit choices are adopted precisely for the sake of preserving this property in the probabilistic case. Taylor expansions have another important property: they are entirely defined by their support. If a simple term s is in the support of the Taylor expansion of a λ-term M, then its coefficient is the inverse of its *multinomial coefficient*, which does not depend on M. Moreover this property is preserved by normalisation. Using explicit choices enforces this result in the probabilistic case, as well.

▶ **Definition 10.** *For any $\sigma \in \Delta_\oplus^{(!)}$ we define the **multinomial coefficient** $\mathrm{m}(\sigma) \in \mathbb{N}$ by:*

$$\mathrm{m}(x) = 1 \qquad \mathrm{m}(\langle s\rangle\ \bar{t}) = \mathrm{m}(s)\mathrm{m}(\bar{t})$$

$$\mathrm{m}(\lambda x.s) = \mathrm{m}(s \oplus_p \bullet) = \mathrm{m}(\bullet \oplus_p s) = \mathrm{m}(s) \qquad \mathrm{m}(\bar{s}) = \prod_{u\in\Delta_\oplus} \bar{s}(u)! \cdot \mathrm{m}(u)^{\bar{s}(u)}$$

where $\bar{s}(u)$ is the multiplicity of u in $\bar{s}$.

▶ **Definition 11.** *A uniform term $S \in \mathbb{R}^+\langle\Delta_\oplus^{(!)}\rangle$ is called **regular** if for all $\sigma \in \mathrm{supp}(S)$, $S_\sigma = \frac{1}{\mathrm{m}(\sigma)}$.*

Multinomial coefficients correspond to the number of permutations of multisets which preserve the description of simple (poly-)terms. For instance, given variables $x_1, \ldots, x_n \in \mathcal{V}$, the coefficient $\mathrm{m}([x_1, \ldots, x_n])$ is exactly the number of permutations $\rho \in \mathfrak{S}_n$ such that $(x_{\rho(1)}, \ldots, x_{\rho(n)}) = (x_1, \ldots, x_n)$. For a more precise interpretation of multinomial coefficients see [9] or [14]. Due to their relation with permutations in multisets, these coefficients appear naturally when we perform substitutions.

▶ **Theorem 12.** *For any $\sigma \in \Delta_\oplus^{(!)}$ uniform, for $x \in \mathcal{V}$, $\bar{t} \in \Delta_\oplus^!$ and $u \in \mathrm{supp}(\delta_x\sigma \cdot \bar{t})$, we have: $(\delta_x\sigma \cdot \bar{t})_u = \frac{\mathrm{m}(\bar{t})\mathrm{m}(\sigma)}{\mathrm{m}(u)}$.*

This theorem ensures that a regular β-redex $\frac{1}{\mathrm{m}(\langle\lambda x.s\rangle\ \bar{t})}.\langle\lambda x.s\rangle\ \bar{t}$ reduces into a regular term. More generally, the theorem is the key step towards proving that regular (poly-)terms always normalise to regular (poly-)terms.

▶ **Theorem 13.** *If $S \in \mathbb{R}^+\langle\Delta_\oplus^{(!)}\rangle$ is regular then $\mathrm{nf}(S)$ is regular.*

2.4 Regularity and the Exponential

The regularity of terms is preserved by the constructors of simple resource terms.

▶ **Proposition 14.** *For all $x \in \mathcal{V}$, $S \in \mathbb{R}^+\langle\Delta_\oplus\rangle$ regular and $\overline{T} \in \mathbb{R}^+\langle\Delta_\oplus^!\rangle$ regular, the terms $1.x$, $\lambda x.S$, $S \oplus_p \bullet$, $\bullet \oplus_p S$ and $\langle S\rangle\ \overline{T}$ are regular.*

One may expect a similar result for poly-terms: if $S_1,\dots,S_n$ in $\mathbb{R}^+\langle\Delta_\oplus\rangle$ are regular then $[S_1,\dots,S_n]$ is regular. However, this is not the case: $1.x$ is regular and yet $1.[x,x]$ is not. Indeed nontrivial coefficients appear in $\mathrm{m}(\sigma)$ precisely when σ contains simple poly-terms with multiplicities greater than 1, so the regular sum with the same support as $[S_1,\dots,S_n]$ has no simple description. A natural way to build regular poly-terms from regular terms is to use the following construction.

▶ **Definition 15.** *The **exponential** of $S \in \mathbb{R}^+\langle\Delta_\oplus\rangle$ is $!S = \sum_{n\in\mathbb{N}} \frac{1}{n!}[S^n] \in \mathbb{R}^+\langle\Delta^!_\oplus\rangle$, where $[S^n]$ stands for the poly-term $[S,\dots,S]$ with n copies of S.*

▶ **Proposition 16.** *If $S \in \mathbb{R}^+\langle\Delta_\oplus\rangle$ is regular then $!S$ is regular.*

Proof. The key point is that the number of sequences $(s_1,\dots,s_n)$ which describe a given simple poly-term $\overline{s} = [s_1,\dots,s_n]$ is exactly $\frac{n!}{\prod_{u\in\Delta_\oplus}\overline{s}(u)!}$. ◀

With these results, we have all the ingredients we need to translate (probabilistic) λ-terms into regular terms: variables and abstractions of regular terms are regular, and we can define an application between regular terms following Girard's call-by-name translation of intuitionistic logic into linear logic [10]: S applied to T is $\langle S\rangle\ !T$.

3 Explicit Probabilistic Taylor Expansion

This section is devoted to defining and studying the *Taylor expansion with explicit choices*, or *explicit Taylor expansion*, of probabilistic λ-terms. It is named as such because its target is the set of probabilistic resource terms, as defined in the previous section, rather than the usual ones. This is *not* the main contribution of this paper, but an intermediate step in the study of Taylor expansion as defined in Section 4.

3.1 The Definition

Probabilistic λ-terms are λ-terms enriched with a probabilistic choice operator.

▶ **Definition 17.** *The set of **probabilistic λ-terms** Λ^+ is:*

$$M, N \in \Lambda^+ := x \mid \lambda x.M \mid M\ N \mid M \oplus_p N$$

▶ **Definition 18.** *The **explicit Taylor expansion** $M^\oplus$ is defined inductively as follows:*

$$x^\oplus = x \qquad (M\ N)^\oplus = \langle M^\oplus\rangle\ !N^\oplus = \sum_{n\in\mathbb{N}} \frac{1}{n!}\langle M^\oplus\rangle\ [(N^\oplus)^n]$$

$$(\lambda x.M)^\oplus = \lambda x.M^\oplus \qquad (M \oplus_p N)^\oplus = (M^\oplus \oplus_p \bullet) + (\bullet \oplus_p N^\oplus)$$

The results from the previous section immediately imply that Taylor expansions are regular resource terms and that they are normalisable.

▶ **Proposition 19.** *For all $M \in \Lambda^+$, the explicit Taylor expansion $M^\oplus$ is uniform and regular.*

Proof. This is a direct consequence of Proposition 14 and Proposition 16. ◀

▶ **Corollary 20.** *Every explicit Taylor expansion $M^\oplus$ has a normal form $\mathrm{nf}(M^\oplus)$, which we call the **explicit Taylor normal form** of M, and which is regular.*

Proof. This is given by Theorem 13. ◀

3.2 Probabilistic Reduction

In the literature, the probabilistic λ-calculus is usually endowed with a labelled transition relation $\xrightarrow{p}$ describing a probabilistic reduction process, where a choice $M \oplus_p N$ reduces to M with probability p and to N with probability $1-p$. Here to emphasise the correspondence between such a reduction and the constructors $s \oplus_p \bullet$ and $\bullet \oplus_p t$ of the resource calculus we rather use labels l, p and r, p to explicit whether we reduce to the left-hand side or the right-hand side of a choice $\oplus_p$. Since we were mostly interested in normalisation in the resource calculus, we will only consider a big-step operational semantics for the probabilistic λ-calculus.

▶ **Definition 21.** ***Head contexts*** *are contexts of the form $\lambda\vec{x}.[\,]\,\vec{P}$, and are indicated with the metavariable H.* ***Head normal forms*** *are terms of the form $H[y]$. We write* hnf *for the set of all head normal forms. We now define a formal system deriving judgements in the form $\rho \vdash M \twoheadrightarrow h$ where $M \in \Lambda^+$, $h \in \mathrm{hnf}$ and ρ is a finite sequence of elements in $\{\mathrm{l},\mathrm{r}\} \times [0,1]$:*

$$\frac{}{\epsilon \vdash h \twoheadrightarrow h} \qquad \frac{\rho \vdash H[M\left[{}^{N}/{}_{x}\right]] \twoheadrightarrow h}{\rho \vdash H[(\lambda x.M)N] \twoheadrightarrow h} \qquad \frac{\rho \vdash H[M] \twoheadrightarrow h}{(\mathrm{l},p)\cdot\rho \vdash H[M \oplus_p N] \twoheadrightarrow h} \qquad \frac{\rho \vdash H[N] \twoheadrightarrow h}{(\mathrm{r},p)\cdot\rho \vdash H[M \oplus_p N] \twoheadrightarrow h}$$

where ϵ is the empty sequence and $(\ell,p)\cdot(\rho_1,\ldots,\rho_n) = ((\ell,p),\rho_1,\ldots,\rho_n)$ for $\ell \in \{\mathrm{l},\mathrm{r}\}$.

▶ **Lemma 22.** *For all $M \in \Lambda^+$ and ρ there is at most one $h \in \mathrm{hnf}$ such that $\rho \vdash M \twoheadrightarrow h$.*

An interesting property of explicit Taylor expansion is that the explicit Taylor normal form of a term M is precisely given by the explicit Taylor normal forms of the head normal forms h of M, as well as the sequences of choices ρ such that $\rho \vdash M \twoheadrightarrow h$.

▶ **Definition 23.** *Given a sequence of choices ρ and $s \in \Delta_\oplus$ we define $\rho \cdot s \in \Delta_\oplus$ by induction on the length of ρ by:*

$$\epsilon \cdot s = s \qquad ((\mathrm{l},p)\cdot\rho)\cdot s = (\rho\cdot s) \oplus_p \bullet \qquad ((\mathrm{r},p)\cdot\rho)\cdot s = \bullet \oplus_p (\rho \cdot s)$$

We extend this definition to $\mathbb{R}^+\langle\Delta_\oplus\rangle$ by linearity.

▶ **Theorem 24.** *Given any $M \in \Lambda^+$,*

$$\mathrm{nf}(M^\oplus) = \sum_{h\in\mathrm{hnf}} \sum_{\rho\vdash M\twoheadrightarrow h} \rho\cdot\mathrm{nf}(h^\oplus).$$

Proof. First observe that these resource terms are regular: Corollary 20 states that $\mathrm{nf}(M^\oplus)$ and the $\mathrm{nf}(h^\oplus)$ are regular (so the $\rho\cdot\mathrm{nf}(h^\oplus)$ are regular too), and if $\rho \vdash M \twoheadrightarrow h$ and $\rho' \vdash M \twoheadrightarrow h'$ then either $\rho = \rho'$ and by Lemma 22 $h = h'$, or $\rho \neq \rho'$ and then $\rho\cdot\mathrm{nf}(h^\oplus)$ and $\rho'\cdot\mathrm{nf}(h'^\oplus)$ have disjoint supports and we can show they are coherent. Thus we only need to prove that these terms have the same supports. On one hand we can prove that for any $s \in \mathrm{supp}(M^\oplus)$ and any $t \in \mathrm{supp}(\mathrm{nf}(s))$ there exist ρ and h such that $\rho \vdash M \twoheadrightarrow h$ and $t \in \mathrm{supp}(\rho\cdot\mathrm{nf}(h^\oplus))$, by reasoning by induction on the size of s. When s has a head β-redex we need to check that in general if $u \in \mathrm{supp}(U^\oplus)$ and $\overline{v} \in \mathrm{supp}(!V^\oplus)$ then $\mathrm{supp}(\delta_x u \cdot \overline{v}) \subset \mathrm{supp}((U\left[{}^{V}/{}_{x}\right])^\oplus)$, which is immediate by induction on U. On the other hand if $\rho \vdash M \twoheadrightarrow h$ we prove that for any $t \in \mathrm{supp}(\mathrm{nf}(h^\oplus))$ we have $\rho\cdot t \in \mathrm{supp}(\mathrm{nf}(M^\oplus))$, by induction on the proof of $\rho \vdash M \twoheadrightarrow h$. ◀

4 Generic Taylor Expansion of Probabilistic λ-terms

4.1 Barycentric Semantics of Choices

The explicit probabilistic Taylor expansion is satisfactory in that it is an extension of deterministic Taylor expansion which preserves its most important properties: it is regular and so are its normal forms. But while deterministic Taylor normal forms are well known to correspond to Böhm trees [7], explicit Taylor normal forms are not such a good denotational semantics for probabilistic λ-calculus, as they take the exact choices made during the reduction into account. For instance the terms $x \oplus_{\frac{1}{2}} y$ and $y \oplus_{\frac{1}{2}} x$ have *distinct* explicit Taylor normal forms while one could expect them to have *the same semantics.* More precisely we expect any model of the probabilistic λ-calculus to interpret probabilistic choices as a *barycentric sum* respecting the following equivalence.

▶ **Definition 25.** *The* ***barycentric equivalence*** $\equiv_{\mathrm{bar}}$ *is the least congruence on* Λ^+ *such that for all* $M, N, P \in \Lambda^+$ *and* $p, q \in [0,1]$*:*

$$M \oplus_p N \equiv_{\mathrm{bar}} N \oplus_{1-p} M \qquad\qquad M \oplus_p M \equiv_{\mathrm{bar}} M$$
$$(M \oplus_p N) \oplus_q P \equiv_{\mathrm{bar}} M \oplus_{pq} (N \oplus_{\frac{q(1-p)}{1-pq}} P) \text{ if } pq \neq 1 \qquad\qquad M \oplus_1 N \equiv_{\mathrm{bar}} M$$

We want a notion of Taylor expansion M^* such that if $M \equiv_{\mathrm{bar}} N$ then $M^* = N^*$. This is easy to achieve, as the resource calculus stemmed precisely from quantitative models of the λ-calculus, and resource terms are linear combinations.

▶ **Definition 26.** *The sets of* ***simple resource terms*** Δ *and of* ***simple resource poly-terms*** $\Delta^!$ *are:*

$$s, t \in \Delta := x \mid \lambda x.s \mid \langle s \rangle\ \bar{t} \qquad\qquad \bar{s}, \bar{t} \in \Delta^! := [s_1, \dots, s_n]$$

The set of ***resource terms*** *is* $\mathbb{R}^+\langle\Delta\rangle$ *and the set of* ***resource poly-terms*** *is* $\mathbb{R}^+\langle\Delta^!_\oplus\rangle$*.*

▶ **Definition 27.** *The* ***Taylor expansion*** $M^* \in \mathbb{R}^+\langle\Delta\rangle$ *of a term* $M \in \Lambda^+$ *is defined inductively as follows:*

$$x^* = x \qquad\qquad (M\ N)^* = \sum_{n \in \mathbb{N}} \frac{1}{n!} \langle M^* \rangle\ [(N^*)^n]$$
$$(\lambda x.M)^* = \lambda x.M^* \qquad\qquad (M \oplus_p N)^* = pM^* + (1-p)N^*$$

The definition of the Taylor expansion of a probabilistic choice immediately gives the expected property.

▶ **Proposition 28.** *If* $M \equiv_{\mathrm{bar}} N$ *then* $M^* = N^*$*.*

4.2 Normalisation

Unfortunately, these Taylor expansions lack all the good properties of explicit expansions: they are not entirely defined by their support, and those supports are not uniform, so we do not even know if such Taylor expansions admit normal forms. But there is actually a close relationship between explicit and non explicit Taylor expansions which can be used to recover our most important results. Indeed, switching from the explicit Taylor expansion to the Taylor expansion simply amounts to using coefficients instead of explicit choices.

▶ **Definition 29.** *Given any $\sigma \in \Delta_{\oplus}^{(!)}$ we define $|\sigma| \in \Delta^{(!)}$ and a probability $\mathcal{P}(\sigma)$ as follows:*

$$|x| = x \qquad \mathcal{P}(x) = 1$$
$$|\lambda x.s| = \lambda x.|s| \qquad \mathcal{P}(\lambda x.s) = \mathcal{P}(s)$$
$$|\langle s\rangle\ \bar{t}| = \langle |s|\rangle\ |\bar{t}| \qquad \mathcal{P}(\langle s\rangle\ \bar{t}) = \mathcal{P}(s)\mathcal{P}(\bar{t})$$
$$|s \oplus_p \bullet| = |s| \qquad \mathcal{P}(s \oplus_p \bullet) = p\mathcal{P}(s)$$
$$|\bullet \oplus_p s| = |s| \qquad \mathcal{P}(\bullet \oplus_p s) = (1-p)\mathcal{P}(s)$$
$$|[s_1, \dots, s_n]| = [|s_1|, \dots, |s_n|] \qquad \mathcal{P}([s_1, \dots, s_n]) = \prod_{i=1}^{n} \mathcal{P}(s_i)$$

To any probabilistic resource (poly-)term $S \in \mathbb{R}^+\langle\Delta_{\oplus}^{(!)}\rangle$ one could associate the resource term $\sum_{\sigma\in\Delta_{\oplus}^{(!)}} S_\sigma \mathcal{P}(\sigma).|\sigma|$. But just like with normalisation, infinite coefficients may appear. For instance, removing the choices from $S = \sum((x \oplus_1 \bullet)\dots) \oplus_1 \bullet$ (the sum of all simple terms with x under any number of left choices) could give x an infinite coefficient. Fortunately, we do not get any infinite coefficient if we work with regular terms.

▶ **Proposition 30.** *For any $\mathcal{S} \subset \Delta_{\oplus}^{(!)}$ such that for all $\sigma, \sigma' \in \mathcal{S}$, $\sigma \frown \sigma'$ and $|\sigma| = |\sigma'|$ we have $\sum_{\sigma\in\mathcal{S}} \mathcal{P}(\sigma) \leq 1$.*

▶ **Corollary 31.** *For all $S \in \mathbb{R}^+\langle\Delta_{\oplus}^{(!)}\rangle$ regular, $\sum_{\sigma\in\Delta_{\oplus}^{(!)}} S_\sigma \mathcal{P}(\sigma).|\sigma|$ is in $\mathbb{R}^+\langle\Delta^{(!)}\rangle$.*

In particular, we can apply this process to explicit Taylor expansions *and to their normal forms*. It is easy to see that we associate to every explicit Taylor expansion the corresponding Taylor expansion, but more interestingly erasing choices commutes with normalisation.

▶ **Proposition 32.** *For any $M \in \Lambda^+$:*

$$\sum_{s\in\Delta_\oplus} M_s^{\oplus}\mathcal{P}(s).|s| = M^* \qquad \sum_{t\in\Delta_\oplus} \mathrm{nf}(M^{\oplus})_t \mathcal{P}(t).|t| = \sum_{s\in\Delta} M_s^*.\mathrm{nf}(s)$$

*hence $\sum_{s\in\Delta} M_s^*.\mathrm{nf}(s)$ is well defined. We denote it by $\mathrm{nf}(M^*)$ and we call it the **Taylor normal form** of M.*

Proof. The key point is that $\mathrm{nf}(|\sigma|) = |\mathrm{nf}(\sigma)|$ and for any $\tau \in \mathrm{supp}(\mathrm{nf}(\sigma))$, $\mathcal{P}(\tau) = \mathcal{P}(\sigma)$. ◀

4.3 Adequacy

The behaviour of a probabilistic λ-term is usually described as a (sub-)probability distribution over the possible results of its evaluation. In particular, the *observable* behaviour of a term is its *convergence probability*, i.e. the probability for its computation to terminate [11, 5]. To show that the Taylor expansion gives a meaningful semantics we will prove it is *adequate*, i.e. it does not equate terms which are not observationally equivalent. We can actually show a more refined result, given as a Corollary of Theorem 24: the Taylor normal form of a term is given by the Taylor normal forms of its head normal forms.

▶ **Definition 33.** *The any sequence of choices ρ we associate a probability $\mathcal{P}(\rho)$ by:*

$$\mathcal{P}(\epsilon) = 1 \qquad \mathcal{P}((\mathrm{l}, p) :: \rho) = p\mathcal{P}(\rho) \qquad \mathcal{P}((\mathrm{r}, p) :: \rho) = (1-p)\mathcal{P}(\rho)$$

*The probability $\mathcal{P}(M \twoheadrightarrow h)$ for $M \in \Lambda^+$ to reduce into a head normal form h and its **convergence probability** $\mathcal{P}_{\Downarrow}(M)$ are defined as follows:*

$$\mathcal{P}(M \twoheadrightarrow h) := \sum_{\rho\vdash M\twoheadrightarrow h} \mathcal{P}(\rho) \qquad \mathcal{P}_{\Downarrow}(M) = \sum_{h\in\mathrm{hnf}} \mathcal{P}(M \twoheadrightarrow h).$$

▶ **Proposition 34.** *For $M \in \Lambda^+$ we have:*

$$\mathrm{nf}(M^*) = \sum_{h \in \mathrm{hnf}} \mathcal{P}\left(M \twoheadrightarrow h\right) \mathrm{nf}(h^*).$$

Proof. This is given by Proposition 32 and Theorem 24. Observe that for any ρ and $s \in \mathrm{nf}(h^{\oplus})$ we have $\mathcal{P}(\rho \cdot s) = \mathcal{P}(\rho)\mathcal{P}(s)$ and $|\rho \cdot s| = |s|$. ◀

The adequacy follows immediately.

▶ **Proposition 35.** *If $\mathrm{nf}(M^*) = \mathrm{nf}(N^*)$ then for all context C, $\mathcal{P}_{\Downarrow}(C[M]) = \mathcal{P}_{\Downarrow}(C[N])$, i.e. M and N are contextually equivalent.*

Proof. First the convergence probability of a term M is exactly the sum of the coefficients $\mathrm{nf}(M^*)_{\lambda\vec{x}.y\,[\,]\ldots[\,]}$. Second if $\mathrm{nf}(M^*) = \mathrm{nf}(N^*)$ then $\mathrm{nf}(C[M]^*) = \mathrm{nf}(C[N]^*)$ for all C. ◀

5 On the Taylor Expansion and Böhm Trees

5.1 A Commutation Theorem

Deterministic Taylor normal forms are an adequate semantics for the probabilistic λ-calculus, but more precisely they are known to correspond to Böhm trees [7]. We are now able to show that this result extends to the probabilistic case.

▶ **Definition 36.** *The sets of* ***probabilistic Böhm trees*** *$\mathcal{PT}_d$ and of* ***probabilistic value trees*** *$\mathcal{VT}_d$ for $d \in \mathbb{N}$ are defined inductively by induction on the* ***depth*** *d:*

$$\mathcal{PT}_0 = \{\bot : \emptyset \to [0,1]\} \qquad \mathcal{VT}_0 = \emptyset$$
$$\mathcal{PT}_{d+1} = \mathbf{D}(\mathcal{VT}_{d+1}) \qquad \mathcal{VT}_{d+1} = \{\lambda\vec{x}.y\ \mathbf{T}_1\ \cdots\ \mathbf{T}_m \mid \mathbf{T}_1, \ldots, \mathbf{T}_m \in \mathcal{PT}_d\}$$

where $\mathbf{D}(X)$ is the set of countable-support subprobability distributions on any set X, $\bot$ is the only subprobability distribution over the empty set, i.e. over $\mathcal{VT}_0$.

▶ **Definition 37.** *We define $PT_d(M)$ for $M \in \Lambda^+$ and $d \geq 0$, and $VT_d(h)$ for $h \in \mathrm{hnf}$ and $d \geq 1$ by induction on the depth d as follows: $PT_0(M)$ is the unique function $\emptyset \to [0,1]$ and*

$$PT_{d+1}(M) = \mathbf{t} \mapsto \sum_{h \in VT_{d+1}^{-1}(\mathbf{t})} \mathcal{P}\left(M \twoheadrightarrow h\right)$$
$$VT_{d+1}(\lambda\vec{x}.y\ M_1\ \ldots\ M_m) = \lambda\vec{x}.y\ PT_d(M_1)\ \ldots\ PT_d(M_m)$$

Intuitively the Böhm tree of a term M would be some limit of its finite Böhm approximants $PT_d(M)$. To avoid making the structure of Böhm trees of infinite depth explicit, we simply write $PT(M)$ for the sequence $(PT_d(M))_{d \in \mathbb{N}}$. In particular we say that M and N have *the same Böhm tree* iff $PT_d(M) = PT_d(N)$ for every $d \in \mathbb{N}$.

The definition of the Taylor expansion can easily be generalised to finite-depth Böhm trees. We simply define $\mathbf{T}^*$ for $\mathbf{T} \in \mathcal{PT}_d$ and $\mathbf{t}^*$ for $\mathbf{t} \in \mathcal{VT}_{d+1}$ by:

$$\mathbf{T}^* = \sum_{\mathbf{t} \in \mathcal{VT}_d} \mathbf{T}(\mathbf{t})\mathbf{t}^* \qquad (\lambda\vec{x}.y\ \mathbf{T}_1\ \ldots\ \mathbf{T}_m)^* = \lambda\vec{x}.\langle y\rangle\ !\mathbf{T}_1^* \ldots !\mathbf{T}_m^*$$

We extend this definition to infinite Böhm trees as follows: if $s \in \Delta$ contains at most d_s layers of nested multisets then for any $M \in \Lambda^+$, $PT_d(M)^*_s = PT_{d_s}(M)^*_s$ for all $d \geq d_s$, so $PT(M)^*_s$ can be taken as $PT_{d_s}(M)^*_s$. Then the Taylor normal form of a term is exactly the Taylor expansion of its Böhm tree.

▶ **Theorem 38.** *For all* $M \in \Lambda^+$, $\mathrm{nf}(M^*) = (PT(M))^*$.

Proof. We prove $\mathrm{nf}(M^*)_s = (PT(M))^*_s$ by induction on d_s, using to Proposition 34. ◀

This theorem is important but it does not actually prove the correspondence between Böhm trees and Taylor expansions: we still do not know if Taylor expansion is injective on Böhm trees. In the deterministic case this is simple to prove: to every deterministic Böhm tree $\mathbf{T}$ of depth d we can associate a simple resource term $s_{\mathbf{T}}$ such that for all $M \in \Lambda$, $BT_d(M) = \mathbf{T}$ iff $s_{\mathbf{T}} \in \mathrm{supp}(\mathrm{nf}(M^*))$ (by associating $\lambda\vec{x}.\langle y\rangle\ [s_{\mathbf{T}_1}] \dots [s_{\mathbf{T}_m}]$ to $\lambda\vec{x}.y\,\mathbf{T}_1 \dots \mathbf{T}_m$). This works because ordinary Böhm trees are not quantitative, thus the quantitative part of their Taylor expansions (the coefficients) is irrelevant. The situation is more complicated in the probabilistic case, as Taylor expansions are no longer defined solely by their supports. The rest of this article is devoted to proving injectivity for the probabilistic Taylor expansion.

5.2 Böhm Tests

In order to better understand coefficients in probabilistic Taylor expansions and to get our injectivity property, we use a notion of *testing* coming from the literature on labelled Markov decision processes [17].

▶ **Definition 39** (Böhm Tests). *The classes of* Böhm term tests *(BTTs) and* Böhm hnf tests *(BHTs) are given as follows, by mutual induction:*

$$T, U ::= \omega \mid T \wedge U \mid \mathsf{ev}(t) \qquad\qquad t, u ::= \omega \mid t \wedge u \mid (\lambda x_1.\cdots.\lambda x_n.y)(T^1, \dots, T^m)$$

The probability of success of a BTT T on a term M and the probability of success of a BHT t on an head-normal-form h, indicated as $\mathsf{Pr}(T, M)$ and $\mathsf{Pr}(t, h)$ respectively, are defined as follows:

$$\mathsf{Pr}(T \wedge U, M) = \mathsf{Pr}(T, M) \cdot \mathsf{Pr}(U, M); \qquad \mathsf{Pr}(\omega, M) = \mathsf{Pr}(\omega, h) = 1;$$

$$\mathsf{Pr}(t \wedge u, h) = \mathsf{Pr}(t, h) \cdot \mathsf{Pr}(u, h); \qquad \mathsf{Pr}(\mathsf{ev}(t), M) = \sum_{h \in \mathrm{hnf}} \mathcal{P}(M \twoheadrightarrow h) \cdot \mathsf{Pr}(t, h);$$

$$\mathsf{Pr}((\lambda x_1.\cdots.\lambda x_n.y)(T^1, \dots, T^m), \lambda x_1 \cdots .\lambda x_n.yM_1 \cdots M_m) = \Pi_{i=1}^m \mathsf{Pr}(T^i, M_i);$$

$$\mathsf{Pr}((\lambda x_1.\cdots.\lambda x_n.y)(T^1, \dots, T^m), h) = 0, \textit{otherwise}$$

The following is the first step towards proving the main result of this paper, as it characterises Böhm tree equality as equality of families of real numbers.

▶ **Theorem 40.** *Two terms M and N have the same Böhm trees iff for every BTT T it holds that* $\mathsf{Pr}(M, T) = \mathsf{Pr}(N, T)$.

A detailed proof of Theorem 40 can be found in the Extended Version of this paper [4]. Let us briefly discuss how the proof goes. The starting point is a result due to van Breugel et al. [17], which establishes a precise correspondence between bisimilarity and testing in a probabilistic scenario: two states s and s' of any labelled Markov decision processes (satisfying certain natural conditions) are bisimilar *iff* the probabilities of any test T to succeed in s and in s' are the same, and tests are defined inductively as follows:

$$T, U ::= \omega \mid T \wedge U \mid a(T)$$

where a is an action of the underlying labelled Markov decision process. Given that Böhm trees can be naturally presented coinductively, most of the involved work has already been done. What remains to be proved, then, is that the result above also holds for transition

systems in which firing an action a brings the system into k distinct states, thus capturing the kind of tree-like evolution typical of Böhm trees. This can be done by translating any such tree-like transition system into a linear one in such a way that bisimilarity and testing-equivalence remain unaltered.

To achieve our main result we now need to relate the probability of success of tests to the coefficients of Taylor normal forms. This is the purpose of Section 6.

6 Implementing Tests as Resource Terms

There is a very tight correspondence between simple resource terms and Böhm tests, but this correspondence does not hold for *all* Böhm tests. Simple resource terms can be seen as a particular class of Böhm tests.

▶ **Definition 41.** *The classes of* resource Böhm term tests *(rBTTs) and* resource Böhm hnf tests *(rBHTs) are given as follows, by mutual induction:*

$$T, U ::= \omega \mid T \wedge U \mid \mathsf{ev}(t) \qquad t ::= (\lambda x_1.\cdots.\lambda x_n.y)(T^1, \ldots, T^m)$$

▶ **Definition 42.** *For every rBTT T we define a simple poly-term $\overline{s}_T$ and for every rBHT t we define a simple term s_t in the following way:*

$$\overline{s}_\omega = [\,] \quad \overline{s}_{T\wedge U} = \overline{s}_T \cdot \overline{s}_U \quad \overline{s}_{\mathsf{ev}(t)} = [s_t] \quad s_{(\lambda\vec{x}.y)(T^1,\ldots,T^m)} = \lambda\vec{x}.\langle y\rangle\ \overline{s}_{T^1}\ \ldots\ \overline{s}_{T^m}$$

The similarity between simple resource terms and resource Böhm tests is more than structural: the probability of success of a resource Böhm test is actually given by a coefficient in the Taylor normal form.

▶ **Proposition 43.**
1. *For every rBTT T and $M \in \Lambda^+$, $!\mathrm{nf}(M^*)_{\overline{s}_T} = \frac{\mathsf{Pr}(T,M)}{\mathrm{m}(\overline{s}_{T_t})}$.*
2. *For every rBHT t and $h \in \mathrm{hnf}$, $\mathrm{nf}(h^*)_{s_t} = \frac{\mathsf{Pr}(t,h)}{\mathrm{m}(s_{T_h})}$.*

Proof. We reason by induction on tests. Observe that these can be considered modulo commutativity and associativity of the conjunction and modulo $\omega\wedge T \simeq T$: these equivalences preserve both the results of testing and the associated simple resource (poly-)terms. Then every rBTT is equivalent either to ω or to a conjunction $T = \mathsf{ev}(t_1)\wedge\cdots\wedge\mathsf{ev}(t_k)$. In the first case we always have $!\mathrm{nf}(M^*)_{[\,]} = 1$. In the second case just like in the proof of regularity of the exponential (Proposition 16) for any $M \in \Lambda^+$ we have $!\mathrm{nf}(M^*)_{\overline{s}_T} = \frac{1}{\prod_{u\in\Delta}\overline{s}_T(u)!}\prod_{i=1}^{k}\mathrm{nf}(M^*)_{s_{t_i}}$. To conclude we want to show that $\mathrm{nf}(M^*)_{s_{t_i}} = \frac{\mathsf{Pr}(\mathsf{ev}(t_i),M)}{\mathrm{m}(s_{t_i})}$ for all $i \leq k$. We have by definition $\mathsf{Pr}(\mathsf{ev}(t_i), M) = \sum_{h\in\mathrm{hnf}} \mathcal{P}\,(M \twoheadrightarrow h)\cdot\mathsf{Pr}(t_i, h)$, and Proposition 34 gives $\mathrm{nf}(M^*)_{s_{t_i}} = \sum_{h\in\mathrm{hnf}}\mathcal{P}\,(M \twoheadrightarrow h)\cdot\mathrm{nf}(h^*)_{s_{t_i}}$, so we conclude by induction hypothesis on t_i. Now given a rBHT $t = (\lambda\vec{x}.y)(T^1,\ldots,T^m)$ and $h \in \mathrm{hnf}$ we have either $\mathrm{nf}(h^*)_{s_t} = \prod_{i=1}^{m} !\mathrm{nf}(M_i^*)_{\overline{s}_{T^i}}$ and $\mathsf{Pr}(t,h) = \prod_{i=1}^{m}\mathsf{Pr}(T^i, M_i)$ if h is of the form $\lambda\vec{x}.y\ M_1\ \ldots\ M_m$, in which case we conclude by induction hypothesis, or $\mathrm{nf}(h^*)_{s_t} = \mathsf{Pr}(t,h) = 0$ otherwise. ◀

With this result, we completely characterise Taylor normal forms by resource Böhm tests.

▶ **Corollary 44.** *Two terms M and N have the same Taylor normal form iff for every rBTT T it holds that $\mathsf{Pr}(M,T) = \mathsf{Pr}(N,T)$.*

Proof. Simply observe that every simple resource term in normal form is equal to s_T for some resource Böhm test T. ◀

Thanks to Theorem 40 and Corollary 44 both Böhm tree equality and Taylor normal form equality are characterised by tests. They still leave a gap in our reasoning, as not all Böhm tests are resource Böhm tests. This difference is not just cosmetic: $\mathsf{ev}(\omega)$ is a valid Böhm test which computes the convergence probability of any λ-term, which cannot be done using only resource Böhm tests. More precisely this cannot be done using a *single* Böhm test. To fill the gap between Böhm tests and resource Böhm tests we observe that any of the former can be simulated by a *family* of resource Böhm tests.

▶ **Proposition 45.** *For every BTT T there is a family $(T_i)_{i\in I}$ of rBTTs of arbitrary size (possibly empty, possibly infinite) such that for all λ-term M we have* $\mathsf{Pr}(T, M) = \sum_{i\in I} \mathsf{Pr}(T_i, M)$.

Proof. We prove this, as well as the corresponding result for BHTs, by induction on the size of tests. In the case of BTTs, the result is simply given by induction hypothesis. To the BTT ω we associate the single-element family (w), to $T \wedge U$ we associate $(T_i \wedge U_j)_{i\in I, j\in J}$ where $(T_i)_{i\in I}$ and $(U_j)_{j\in J}$ are given by induction hypothesis on T and U, and to $\mathsf{ev}(t)$ we associate $(\mathsf{ev}(t_i))_{i\in I}$. The interesting part of the proof is on BHTs, where we want to remove two constructors. Modulo commutativity and associativity of the conjunction and the equivalence $\omega \wedge T \simeq T$, every BHT is either ω or of the form $(\lambda x_1...x_{n_1}.y_1)(T_1^1, \ldots, T_1^{m_1}) \wedge \cdots \wedge (\lambda x_1...x_{n_k}.y_k)(T_k^1, \ldots, T_k^{m_k})$ with $k \geq 1$. In the first case to ω we associate the family $((\lambda x_1 \ldots x_n.y)(\omega^m))_{m,n\in\mathbb{N}, y\in\mathcal{V}}$ where ω^m denotes the sequence $\omega, \ldots, \omega$ of length m. In the second case if $m_i \neq m_j$, $n_i \neq n_j$ or $y_i \neq y_j$ for some $i, j \leq k$ then the result of the test is always 0, which is simulated by the empty family of rBHTs. Otherwise let $m = m_1$, $n = n_1$ and $y = y_1$, the test is equivalent to $(\lambda x_1 \ldots x_n.y)(T_1^1 \wedge \cdots \wedge T_k^1, \ldots, T_1^m \wedge \cdots \wedge T_k^m)$. We apply the induction hypothesis to the BTTs $T_1^i \wedge \cdots \wedge T_k^i$ to get families $(U_j^i)_{j\in J_i}$ and we associate the family $((\lambda x_1 \ldots x_n.y)(U_{j_1}^1, \ldots, U_{j_m}^m))_{j_1\in J_1,\ldots,j_m\in J_m}$ to the original BHT. ◀

▶ **Corollary 46.** *Given two terms M and N, for every BTT T it holds that* $\mathsf{Pr}(M, T) = \mathsf{Pr}(N, T)$ *iff for every rBTT T it holds that* $\mathsf{Pr}(M, T) = \mathsf{Pr}(N, T)$.

We can now state the main result of this paper.

▶ **Theorem 47.** *Two terms have the same Böhm trees iff their Taylor expansions have the same normal forms.*

Proof. The result follows from Theorem 40, Corollary 46 and Corollary 44. ◀

7 Conclusion

In this paper, we attack the problem of extending the Taylor Expansion construction to the probabilistic λ-calculus, at the same time preserving its nice properties. What we find remarkable about the defined notion of Taylor expansion is that its codomain is the set of *ordinary* resource terms, and that the equivalence induced by the Taylor expansion is precisely the one induced by Böhm trees [13]. The latter, not admitting η, is strictly included in contextual equivalence.

Among the many questions this work leaves open, we could cite the extension of the proposed definition to call-by-value reduction, along the lines of [12], and a formal comparison between the notion of equivalence introduced here and the the one from [15] in which, however, the target language is not the one of ordinary resource terms, but one specifically designed around probabilistic effects.

References

1. H.P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. Studies in Logic and the Foundations of Mathematics. Elsevier Science, 1984.
2. Johannes Borgström, Ugo Dal Lago, Andrew D. Gordon, and Marcin Szymczak. A lambda-calculus foundation for universal probabilistic programming. In *Proc. of ICFP 2016*, pages 33–46, 2016.
3. Gérard Boudol. The Lambda-Calculus with Multiplicities. Technical Report 2025, INRIA Sophia-Antipolis, 1993.
4. Ugo Dal Lago and Thomas Leventis. On the Taylor Expansion of Probabilistic Lambda Terms (Long Version), 2019. `arXiv:1904.09650`.
5. Thomas Ehrhard, Michele Pagani, and Christine Tasson. Full Abstraction for Probabilistic PCF. *J. ACM*, 65(4):23:1–23:44, 2018.
6. Thomas Ehrhard and Laurent Regnier. The differential lambda-calculus. *Theor. Comput. Sci.*, 309(1-3):1–41, 2003.
7. Thomas Ehrhard and Laurent Regnier. Böhm Trees, Krivine's Machine and the Taylor Expansion of Lambda-Terms. In *Proc. of CIE 2006*, pages 186–197, 2006.
8. Thomas Ehrhard and Laurent Regnier. Differential interaction nets. *Theor. Comput. Sci.*, 364(2):166–195, 2006.
9. Thomas Ehrhard and Laurent Regnier. Uniformity and the Taylor expansion of ordinary lambda-terms. *Theor. Comput. Sci.*, 403(2-3):347–372, 2008.
10. Jean-Yves Girard. Linear Logic. *Theor. Comput. Sci.*, 50:1–102, 1987.
11. Claire Jones and Gordon D. Plotkin. A Probabilistic Powerdomain of Evaluations. In *Proc. of LICS 1989*, pages 186–195, 1989.
12. Emma Kerinec, Giulio Manzonetto, and Michele Pagani. Revisiting Call-by-value Bohm trees in light of their Taylor expansion, 2018. `arXiv:1809.02659`.
13. Thomas Leventis. Probabilistic Böhm Trees and Probabilistic Separation. In *Proc. of LICS 2018*, pages 649–658, 2018.
14. Takeshi Tsukada, Kazuyuki Asada, and C.-H. Luke Ong. Generalised species of rigid resource terms. In *Proc. of LICS 2017*, pages 1–12, 2017.
15. Takeshi Tsukada, Kazuyuki Asada, and C.-H. Luke Ong. Species, Profunctors and Taylor Expansion Weighted by SMCC: A Unified Framework for Modelling Nondeterministic, Probabilistic and Quantum Programs. In *Proc. of LICS 2018*, pages 889–898, 2018.
16. Matthijs Vákár, Ohad Kammar, and Sam Staton. A domain theory for statistical probabilistic programming. *PACMPL*, 3(POPL):36:1–36:29, 2019.
17. Franck van Breugel, Michael W. Mislove, Joël Ouaknine, and James Worrell. Domain theory, testing and simulation for labelled Markov processes. *Theor. Comput. Sci.*, 333(1-2):171–197, 2005.
18. Lionel Vaux. The algebraic lambda calculus. *Mathematical Structures in Computer Science*, 19(5):1029–1059, 2009.
19. Lionel Vaux Auclair and Federico Olimpieri. On the Taylor expansion of λ-terms and the groupoid structure of their rigid approximants. Informal proc. of TLLA 2018, 2018.

Proof Normalisation in a Logic Identifying Isomorphic Propositions

Alejandro Díaz-Caro
Instituto de Ciencias de la Computación (CONICET-Universidad de Buenos Aires), Ciudad Autónoma de Buenos Aires, Argentina
Universidad Nacional de Quilmes, Bernal (Buenos Aires), Argentina
https://www-2.dc.uba.ar/staff/adiazcaro/
adiazcaro@icc.fcen.uba.ar

Gilles Dowek
Inria, LSV, ENS Paris-Saclay, France
http://www.lsv.fr/~dowek/
gilles.dowek@ens-paris-saclay.fr

Abstract

We define a fragment of propositional logic where isomorphic propositions, such as $A \wedge B$ and $B \wedge A$, or $A \Rightarrow (B \wedge C)$ and $(A \Rightarrow B) \wedge (A \Rightarrow C)$ are identified. We define System I, a proof language for this logic, and prove its normalisation and consistency.

2012 ACM Subject Classification Theory of computation → Proof theory; Theory of computation → Type theory; Mathematics of computing → Lambda calculus

Keywords and phrases Simply typed lambda calculus, Isomorphisms, Logic, Cut-elimination, Proof-reduction

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.14

Related Version A variant of the system presented in this paper, has been published in https://arxiv.org/abs/1303.7334v1 (LSFA 2012 [13]), with a sketch of a subject reduction proof but no normalisation or consistency proofs.

Funding Partially supported by ECOS-Sud A17C03, PICT 2015-1208, and the French-Argentinian laboratory SINFIN.

1 Introduction

1.1 Identifying isomorphic propositions

In mathematics, addition is associative and commutative, multiplication distributes over addition, etc. In contrast, in logic conjunction is neither associative nor commutative, implication does not distribute over conjunction, etc. For instance, the propositions $A \wedge B$ and $B \wedge A$ are different: if $A \wedge B$ has a proof, then so does $B \wedge A$, but if r is a proof of $A \wedge B$, then it is not a proof of $B \wedge A$.

A first step towards considering $A \wedge B$ and $B \wedge A$ as the same proposition has been made in [6, 11, 12, 26], where a notion of isomorphic propositions has been defined: two propositions A and B are isomorphic if there exist two proofs of $A \Rightarrow B$ and $B \Rightarrow A$ whose composition, in both ways, is the identity.

For the fragment of propositional logic restricted to the operations $\Rightarrow$ and $\wedge$, all the isomorphisms are consequences of the following four:

$$A \wedge B \equiv B \wedge A \tag{1}$$
$$A \wedge (B \wedge C) \equiv (A \wedge B) \wedge C \tag{2}$$
$$A \Rightarrow (B \wedge C) \equiv (A \Rightarrow B) \wedge (A \Rightarrow C) \tag{3}$$
$$(A \wedge B) \Rightarrow C \equiv A \Rightarrow B \Rightarrow C \tag{4}$$

For example, $(A \Rightarrow B \Rightarrow C) \equiv (B \Rightarrow A \Rightarrow C)$ is a consequence of (4) and (1) [6].

© Alejandro Díaz-Caro and Gilles Dowek;
licensed under Creative Commons License CC-BY
4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019).
Editor: Herman Geuvers; Article No. 14; pp. 14:1–14:23
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper, we go one step further and define a proof language, System I, for the fragment $\Rightarrow$, $\wedge$, such that when $A \equiv B$, then any proof of A is also a proof of B, so the propositions $A \wedge B$ and $B \wedge A$, for instance, are really identical, as they have the same proofs.

The idea of identifying some propositions has already been investigated, for example, in Martin-Löf's type theory [23], in the Calculus of Constructions [8], and in Deduction modulo theory [17, 19], where definitionally equivalent propositions, for instance $A \subseteq B$, $A \in \mathcal{P}(B)$, and $\forall x\ (x \in A \Rightarrow x \in B)$ can be identified. But definitional equality does not handle isomorphisms. For example, $A \wedge B$ and $B \wedge A$ are not identified in these logics. Beside definitional equality, identifying isomorphic types in type theory, is also a goal of the univalence axiom [27].

Isomorphisms make proofs more natural. For instance, to prove $(A \wedge (A \Rightarrow B)) \Rightarrow B$ in natural deduction we need to introduce conjunctive hypothesis $A \wedge (A \Rightarrow B)$ which has to be decomposed into A and $A \Rightarrow B$, while using the isomorphism (4) allows to transform the goal to $A \Rightarrow (A \Rightarrow B) \Rightarrow B$ and introduce directly the hypotheses A and $A \Rightarrow B$, eliminating completely the need for conjunctive hypotheses.

1.2 Lambda-calculus

The proof-language of the fragment of propositional logic restricted to the operations $\Rightarrow$ and $\wedge$ is simply typed lambda-calculus extended with Cartesian product. So, System I is an extension of this calculus where, for example, a pair of functions $\langle r, s\rangle$ of type $(A \Rightarrow B) \wedge (A \Rightarrow C) \equiv A \Rightarrow (B \wedge C)$ can be applied to an argument t of type A, yielding a term $\langle r, s\rangle t$ of type $B \wedge C$. For example, the term $\langle \lambda x^\tau.x, \lambda x^\tau.x\rangle y$ has type $\tau \wedge \tau$. With the usual reduction rules of lambda calculus with pairs, such a term would be normal, but we can also extend the reduction relation, with an equation $\langle r, s\rangle t \rightleftarrows \langle rt, st\rangle$, such that this term is equivalent to $\langle (\lambda x^\tau.x)y, (\lambda x^\tau.x)y\rangle$ and thus reduces to $\langle y, y\rangle$. Taking too many of such equations may lead to non termination (Section 8.1), and taking too few multiplies undesired normal forms. The choice of the rules in this paper is motivated by the goal to have both termination of reduction (Section 5) and consistency (Section 6), that is, no normal closed term of atomic types.

To stress the associativity and commutativity of the notion of pair, we write $r \times s$ instead of $\langle r, s\rangle$ and thus write this equivalence as

$$(r \times s)t \rightleftarrows rt \times st$$

Several similar equivalence rules on terms are introduced: one related to the isomorphism (1), the commutativity of the conjunction, $r \times s \rightleftarrows s \times r$; one related to the isomorphism (2), the associativity of the conjunction, $(r \times s) \times t \rightleftarrows r \times (s \times t)$; two to the isomorphism (3), the distributivity of implication with respect to conjunction, $\lambda x.(r \times s) \rightleftarrows \lambda x.r \times \lambda x.s$ and $(r \times s)t \rightleftarrows rt \times st$; and one related to the isomorphism (4), the currification, $rst \rightleftarrows r(s \times t)$.

One of the difficulties in the design of System I is the design of the elimination rule for the conjunction. A rule like "if $r : A \wedge B$ then $\pi_1(r) : A$", would not be consistent. Indeed, if A and B are two arbitrary types, s a term of type A and t a term of type B, then $s \times t$ has both type $A \wedge B$ and type $B \wedge A$, thus $\pi_1(s \times t)$ would have both type A and type B. A solution is to consider explicitly typed (Church style) terms, and parametrise the projection by the type: if $r : A \wedge B$ then $\pi_A(r) : A$ and the reduction rule is then that $\pi_A(s \times t)$ reduces to s if s has type A.

This rule makes reduction non-deterministic. Indeed, in the particular case where A happens to be equal to B, then both s and t have type A and $\pi_A(s \times t)$ reduces both to s and to t. Notice that, although this reduction rule is non-deterministic, it preserves typing, like the calculus developed in [18], where the reduction is non-deterministic, but verifies subject reduction.

1.3 Non-determinism

Therefore, System I is one of the many non-deterministic calculi in the sense, for instance, of [5,7,9,10,24] and our pair-construction operator $\times$ is also the parallel composition operator of a non-deterministic calculus.

In non-deterministic calculi, the non-deterministic choice is such that if r and s are two λ-terms, the term $r \oplus s$ represents the computation that runs either r or s non-deterministically, that is such that $(r \oplus s)t$ reduces either to rt or st. On the other hand, the parallel composition operator $|$ is such that the term $(r \mid s)t$ reduces to $rt \mid st$ and continue running both rt and st in parallel. In our case, given r and s of type $A \Rightarrow B$ and t of type A, the term $\pi_B((r \times s)t)$ is equivalent to $\pi_B(rt \times st)$, which reduces to rt or st, while the term $rt \times st$ itself would run both computations in parallel. Hence, our $\times$ is equivalent to the parallel composition while the non-deterministic choice $\oplus$ is decomposed into $\times$ followed by π.

In System I, the non-determinism comes from the interaction of two operators, $\times$ and π. This is similar to quantum computing where the non-determinism comes from the interaction of two operators, the fist allowing to build a superposition, that is a linear combination, of two terms $\alpha.r + \beta.t$, and the measurement operator π. In addition, in such calculi, the distributivity rule $(r + s)t \rightleftarrows rt + st$ is seen as the point-wise definition of the sum of two functions.

More generally, the calculus developed in this paper is also related to the algebraic calculi [1–4,14,16,28], some of which have been designed to express quantum algorithms. There is a clear link between the pair constructor $\times$ and the projection π, with the superposition constructor $+$ and the measurement π on these calculi. In these cases, the pair $s + t$ is not interpreted as a non-deterministic choice, but as a superposition of two processes running s and t, and the operator π is the projection related to the measurement, which is the only non-deterministic operator.

Outline

In Section 2, we define the notion of type isomorphism and prove elementary properties of this relation. In Section 3, we introduce System I. In Section 4, we prove its subject reduction. In Section 5, we prove its strong normalisation. In Section 6, we prove its consistency. Finally, in Section 7, we discuss how System I could be used as a programming language.

2 Type isomorphisms

2.1 Types and isomorphisms

Types are defined by the following grammar

$$A, B, C, \dots \quad ::= \quad \tau \mid A \Rightarrow B \mid A \wedge B$$

where τ is the only atomic type.

▶ **Definition 2.1** (Size of a type). *The size of a type is defined as usual by*

$$s(\tau) = 1$$
$$s(A \Rightarrow B) = s(A) + s(B) + 1$$
$$s(A \wedge B) = s(A) + s(B) + 1$$

▶ **Definition 2.2** (Congruence). *The isomorphisms* (1), (2), (3), *and* (4) *define a congruence on types.*

$$A \wedge B \equiv B \wedge A \tag{1}$$
$$A \wedge (B \wedge C) \equiv (A \wedge B) \wedge C \tag{2}$$
$$A \Rightarrow (B \wedge C) \equiv (A \Rightarrow B) \wedge (A \Rightarrow C) \tag{3}$$
$$(A \wedge B) \Rightarrow C \equiv A \Rightarrow B \Rightarrow C \tag{4}$$

2.2 Prime factors

▶ **Definition 2.3** (Prime types). *A prime type is a type of the form $C_1 \Rightarrow \cdots \Rightarrow C_n \Rightarrow \tau$, with $n \geq 0$.*

A prime type is equivalent to $(\bigwedge_{i=1}^n C_i) \Rightarrow \tau$, which is either equivalent to τ or to $C \Rightarrow \tau$, for some C. For uniformity, we may write $\emptyset \Rightarrow \tau$ for τ.

We now show that each type can be decomposed into a conjunction of prime types. We use the notation $[A_i]_{i=1}^n$ for the multiset whose elements are $A_1, \ldots, A_n$. We may write $[A_i]_i$ when the number of elements is not important. If $R = [A_i]_i$ is a multiset of types, then $\mathsf{conj}(R) = \bigwedge_i A_i$.

▶ **Definition 2.4.** *We write $[A_1, \ldots, A_n] \sim [B_1, \ldots, B_m]$ if $n = m$ and $B_i \equiv A_i$.*

▶ **Definition 2.5** (Prime factors). *The multiset of prime factors of a type A is inductively defined as follows*

$$\mathsf{PF}(\tau) = [\tau]$$
$$\mathsf{PF}(A \Rightarrow B) = [(A \wedge C_i) \Rightarrow \tau]_{i=1}^n \quad \textit{where } [C_i \Rightarrow \tau]_{i=1}^n = \mathsf{PF}(B)$$
$$\mathsf{PF}(A \wedge B) = \mathsf{PF}(A) \uplus \mathsf{PF}(B)$$

with the convention that $A \wedge \emptyset = A$.

Note that if $B \Rightarrow \tau \in \mathsf{PF}(A)$, then $s(B) < s(A)$.

▶ **Lemma 2.6.** *For all A, $A \equiv \mathsf{conj}(\mathsf{PF}(A))$.*

Proof. By induction on $s(A)$.

- If $A = \tau$, then $\mathsf{PF}(\tau) = [\tau]$, and so $\mathsf{conj}(\mathsf{PF}(\tau)) = \tau$.
- If $A = B \Rightarrow C$, then $\mathsf{PF}(A) = [(B \wedge C_i) \Rightarrow \tau]_i$, where $[C_i \Rightarrow \tau]_i = \mathsf{PF}(C)$. By the induction hypothesis, $C \equiv \bigwedge_i (C_i \Rightarrow \tau)$, hence, $A = B \Rightarrow C \equiv B \Rightarrow \bigwedge_i (C_i \Rightarrow \tau) \equiv \bigwedge_i (B \Rightarrow C_i \Rightarrow \tau) \equiv \bigwedge_i ((B \wedge C_i) \Rightarrow \tau)$.
- If $A = B \wedge C$, then $\mathsf{PF}(A) = \mathsf{PF}(B) \uplus \mathsf{PF}(C)$. By the induction hypothesis, $B \equiv \mathsf{conj}(\mathsf{PF}(B))$, and $C \equiv \mathsf{conj}(\mathsf{PF}(C))$. Therefore, $A = B \wedge C \equiv \mathsf{conj}(\mathsf{PF}(B)) \wedge \mathsf{conj}(\mathsf{PF}(C)) \equiv \mathsf{conj}(\mathsf{PF}(B \wedge C)) \equiv \mathsf{conj}(\mathsf{PF}(B) \uplus \mathsf{PF}(C)) = \mathsf{conj}(\mathsf{PF}(A))$. ◀

▶ **Lemma 2.7.** *If $A \equiv B$, then $\mathsf{PF}(A) \sim \mathsf{PF}(B)$.*

Proof. First we check that $\mathsf{PF}(A \wedge B) \sim \mathsf{PF}(B \wedge A)$ and similar for the other three isomorphisms. Then we prove by structural induction that if A and B are equivalent in one step, then $\mathsf{PF}(A) \sim \mathsf{PF}(B)$. We conclude by an induction on the length of the derivation of the equivalence $A \equiv B$. ◀

2.3 Measure of types

The size of a type is not preserved by equivalence. For instance, $\tau \Rightarrow (\tau \wedge \tau) \equiv (\tau \Rightarrow \tau) \wedge (\tau \Rightarrow \tau)$, but $s(\tau \Rightarrow (\tau \wedge \tau)) = 5$ and $s((\tau \Rightarrow \tau) \wedge (\tau \Rightarrow \tau)) = 7$. Thus, we define another notion of measure of a type.

▶ **Definition 2.8** (Measure of a type). *The measure of a type is defined as follows*

$$m(A) = \sum_i (m(C_i) + 1) \qquad \textit{where } [C_i \Rightarrow \tau]_i = \mathsf{PF}(A)$$

with the convention that $m(\emptyset) = 0$.

▶ **Lemma 2.9.** *If* $A \equiv B$*, then* $m(A) = m(B)$.

Proof. By induction on $s(A)$. Let $\mathsf{PF}(A) = [C_i \Rightarrow \tau]_i$ and $\mathsf{PF}(B) = [D_j \Rightarrow \tau]_j$. By Lemma 2.7, $[C_i \Rightarrow \tau]_i \sim [D_i \Rightarrow \tau]_i$. Without lost of generality, take $C_i \equiv D_i$. By the induction hypothesis, $m(C_i) = m(D_i)$. Then, $m(A) = \sum_i (m(C_i) + 1) = \sum_i (m(D_i) + 1) = m(B)$. ◀

The following lemma shows that the measure $m(A)$ verifies the usual properties.

▶ **Lemma 2.10.**
1. $m(A \wedge B) > m(A)$
2. $m(A \Rightarrow B) > m(A)$
3. $m(A \Rightarrow B) > m(B)$

Proof.
1. $\mathsf{PF}(A)$ is a strict submultiset of $\mathsf{PF}(A \wedge B)$.
2. Let $\mathsf{PF}(B) = [C_i \Rightarrow \tau]_{i=1}^n$. Then, $\mathsf{PF}(A \Rightarrow B) = [(A \wedge C_i) \Rightarrow \tau]_{i=1}^n$. Hence, $m(A \Rightarrow B) \geq m(A \wedge C_1) + 1 > m(A \wedge C_1) \geq m(A)$.
3. $m(A \Rightarrow B) = \sum_i m(A \wedge C_i) + 1 > \sum_i m(C_i) + 1 = m(B)$. ◀

2.4 Decomposition properties on types

In simply typed lambda calculus, the implication and the conjunction are constructors, that is $A \Rightarrow B$ is never equal to $C \wedge D$, if $A \Rightarrow B = A' \Rightarrow B'$, then $A = A'$ and $B = B'$, and the same holds for the conjunction. This is not the case in System I, where $\tau \Rightarrow (\tau \wedge \tau) \equiv (\tau \Rightarrow \tau) \wedge (\tau \Rightarrow \tau)$, but the connectors still have some coherence properties:
- If $A \Rightarrow B \equiv \bigwedge_{i=1}^n C_i$, then each C_i is equivalent to an implication $A \Rightarrow B_i$, where the conjunction of the B_i is equivalent to B.
- If $A \wedge B \equiv \bigwedge_i C_i$, then each C_i is a conjunction of elements, possibly empty, that contribute to A and to B.

We state these properties in Corollary 2.12 and Lemma 2.15.

▶ **Lemma 2.11.** *If* $A \Rightarrow B \equiv C_1 \wedge C_2$*, then* $C_1 \equiv A \Rightarrow B_1$ *and* $C_2 \equiv A \Rightarrow B_2$ *where* $B \equiv B_1 \wedge B_2$.

Proof. By Lemma 2.7, $\mathsf{PF}(A \Rightarrow B) \sim \mathsf{PF}(C_1 \wedge C_2) = \mathsf{PF}(C_1) \uplus \mathsf{PF}(C_2)$. Let $\mathsf{PF}(B) = [D_i \Rightarrow \tau]_{i=1}^n$, so $\mathsf{PF}(A \Rightarrow B) = [(A \wedge D_i) \Rightarrow \tau]_{i=1}^n$. Without lost of generality, take $\mathsf{PF}(C_1) \sim [(A \wedge D_i) \Rightarrow \tau]_{i=1}^k$ and $\mathsf{PF}(C_2) \sim [(A \wedge D_i) \Rightarrow \tau]_{i=k+1}^n$. Therefore, by Lemma 2.6, we have $A \Rightarrow B \equiv \bigwedge_{i=1}^k ((A \wedge D_i) \Rightarrow \tau) \wedge \bigwedge_{i=k+1}^n ((A \wedge D_i) \Rightarrow \tau) \equiv (A \Rightarrow \bigwedge_{i=1}^k (D_i \Rightarrow \tau)) \wedge (A \Rightarrow \bigwedge_{i=k+1}^n (D_i \Rightarrow \tau))$. Take $B_1 = \bigwedge_{i=1}^k D_i \Rightarrow \tau$ and $B_2 = \bigwedge_{i=k+1}^n D_i \Rightarrow \tau$. Remark that $C_1 \equiv A \Rightarrow B_1$, $C_2 \equiv A \Rightarrow B_2$ and $B \equiv B_1 \wedge B_2$. ◀

	T	U
R	V	X
S	W	Y

	T_1	T_2		T_n
R	V_1	V_2	…………	V_n
S	W_1	W_2	…………	W_n

Figure 1 Lemma 2.13.

▶ **Corollary 2.12.** *If $A \Rightarrow B \equiv \bigwedge_{i=1}^n C_i$, then for $i \in \{1, \dots, n\}$, we have $C_i \equiv A \Rightarrow B_i$ where $B \equiv \bigwedge_{i=1}^n B_i$.*

Proof. By induction on n. By Lemma 2.11, $\bigwedge_{i=1}^{n-1} C_i \equiv A \Rightarrow B'$ and $C_n \equiv A \Rightarrow B_n$, with $B \equiv B' \wedge B_n$. By the induction hypothesis, for $i \leq n-1$, $C_i \equiv A \Rightarrow B_i$ where $B' \equiv \bigwedge_{i=1}^{n-1} B_i$. Hence, $B \equiv \bigwedge_{i=1}^n B_i$. ◀

▶ **Lemma 2.13.** *Let R, S, T and U be four multisets such that $R \uplus S = T \uplus U$, then there exist four multisets V, W, X, and Y such that $R = V \uplus X$, $S = W \uplus Y$, $T = V \uplus W$, and $U = X \uplus Y$, cf. Figure 1.*

Proof. Consider an element $a \in R \uplus S = T \uplus U$. Let r be the multiplicity of a in R, s its multiplicity in S, t its multiplicity in T, and u its multiplicity in U. We have $r + s = t + u$. If $r \leq t$ we put r copies of a in V, $t - r$ in W, 0 in X, and u in Y. Otherwise, we put t in V, 0 in W, $r - t$ in X, and s in Y. ◀

▶ **Corollary 2.14.** *Let R and S be two multisets and $(T_i)_{i=1}^n$ be a family of multisets, such that $R \uplus S = \biguplus_{i=1}^n T_i$. Then, there exist multisets $V_1, \dots, V_n, W_1, \dots, W_n$ such that $R = \biguplus_i V_i$ and $S = \biguplus_i W_i$ and for each i, $T_i = V_i \uplus W_i$, cf. Figure 1.*

Proof. By induction on n. We have $R \uplus S = \biguplus_{i=1}^{n-1} T_i \uplus T_n$. Then, by Lemma 2.13, there exist R', S', V_n, W_n such that $R = R' \uplus V_n$, $S = S' \uplus W_n$, $\biguplus_{i=1}^{n-1} T_i = R' \uplus S'$, and $T_n = V_n \uplus W_n$. By induction hypothesis, there exist $V_1, \dots, V_{n-1}$ and $W_1, \dots, W_{n-1}$ such that $R' = \biguplus_{i=1}^{n-1} V_i$, $S' = \biguplus_{i=1}^{n-1} W_i$ and each $T_i = V_i \uplus W_i$. Hence, $R = \biguplus_{i=1}^n V_i$ and $S = \biguplus_{i=1}^n W_i$. ◀

▶ **Lemma 2.15.** *If $A \wedge B \equiv \bigwedge_{i=1}^n C_i$ then there exists a partition $E \uplus F \uplus G$ of $\{1, \dots, n\}$ such that*
- *$C_i = A_i \wedge B_i$, when $i \in E$;*
- *$C_i = A_i$, when $i \in F$;*
- *$C_i = B_i$, when $i \in G$;*
- *$A = \bigwedge_{i \in E \uplus F} A_i$; and*
- *$B = \bigwedge_{i \in E \uplus G} B_i$.* ◀

Proof. Let $R = \mathsf{PF}(A)$, $S = \mathsf{PF}(B)$, and $T_i = \mathsf{PF}(C_i)$. By Lemma 2.7, we have $\mathsf{PF}(A \wedge B) \sim \mathsf{PF}(\bigwedge_i C_i)$, that is $R \uplus S \sim \biguplus_i T_i$. By Corollary 2.14, there exist V_i and W_i such that $R = \biguplus_{i=1}^n V_i$, $S = \biguplus_{i=1}^n W_i$, and $T_i \sim V_i \uplus W_i$. As T_i is non-empty, V_i and W_i cannot be both empty.
- If V_i and W_i are both non-empty, we let $i \in E$ and $A_i = \mathsf{conj}(V_i)$ and $B_i = \mathsf{conj}(W_i)$. By Lemma 2.6, $C_i \equiv \mathsf{conj}(T_i) \equiv \mathsf{conj}(V_i \uplus W_i) \equiv A_i \wedge B_i$.
- If V_i is non-empty and W_i is empty, we let $i \in F$, and $A_i = \mathsf{conj}(V_i) \equiv \mathsf{conj}(T_i) \equiv C_i$.
- If W_i is non-empty and V_i is empty, we let $i \in G$, and $B_i = \mathsf{conj}(W_i) \equiv \mathsf{conj}(T_i) \equiv C_i$.

As $V_i = \emptyset$ when $i \in G$, we have $A \equiv \mathsf{conj}(R) \equiv \mathsf{conj}(\biguplus_{i \in E \uplus F} V_i) \equiv \bigwedge_{i \in E \uplus F} A_i$.
As $W_i = \emptyset$ when $i \in F$, we have $B \equiv \mathsf{conj}(S) \equiv \mathsf{conj}(\biguplus_{i \in E \uplus G} W_i) \equiv \bigwedge_{i \in E \uplus G} B_i$. ◀

Table 1 The type system.

$$[x\in\mathcal{V}_A]\frac{}{x:A}\ (ax) \qquad [A\equiv B]\frac{r:A}{r:B}\ (\equiv)$$

$$\frac{r:B}{\lambda x^A.r:A\Rightarrow B}\ (\Rightarrow_i) \qquad \frac{r:A\Rightarrow B \quad s:A}{rs:B}\ (\Rightarrow_e) \qquad \frac{r:A \quad s:B}{r\times s:A\wedge B}\ (\wedge_i) \qquad \frac{r:A\wedge B}{\pi_A(r):A}\ (\wedge_e)$$

Table 2 Symmetric relation.

$r\times s\rightleftarrows s\times r$	(COMM)
$(r\times s)\times t\rightleftarrows r\times(s\times t)$	(ASSO)
$\lambda x^A.(r\times s)\rightleftarrows\lambda x^A.r\times\lambda x^A.s$	(DIST$_\lambda$)
$(r\times s)t\rightleftarrows rt\times st$	(DIST$_{\mathsf{app}}$)
$rst\rightleftarrows r(s\times t)$	(CURRY)

3 System I

3.1 Syntax

We associate to each (up to equivalence) prime type A an infinite set of variables $\mathcal{V}_A$ such that if $A\equiv B$ then $\mathcal{V}_A=\mathcal{V}_B$ and if $A\not\equiv B$ then $\mathcal{V}_A\cap\mathcal{V}_B=\emptyset$. The set of terms is defined inductively by the grammar

$$r,s,t,\dots \quad ::= \quad x \mid \lambda x.r \mid rs \mid r\times s \mid \pi_A(r)$$

We recall the type on binding occurrences of variables and write $\lambda x^A.t$ for $\lambda x.t$ when $x\in\mathcal{V}_A$. α-equivalence and substitution are defined as usual. The type system is given in Table 1. We use a presentation of typing rules without explicit context following [21, 25], hence the typing judgments have the form $r:A$. The preconditions of a typing rule is written on its left.

3.2 Operational semantics

The operational semantics of the calculus is defined by two relations: an equivalence relation, and a reduction relation.

▶ **Definition 3.1.** *The symmetric relation $\rightleftarrows$ is the smallest contextually closed relation defined by the rules given in Table 2.*

Each isomorphism induces an equivalence between terms. Two rules however correspond to the isomorphism (3), depending on which distribution is taken into account: elimination or introduction of implication. We write $\rightleftarrows^*$ for the transitive and reflexive closure of $\rightleftarrows$. Note that $\rightleftarrows^*$ is an equivalence relation.

Because of the associativity property of $\times$, the term $r\times(s\times t)$ is equivalent to the term $(r\times s)\times t$, so we can just write it $r\times s\times t$.

As explained in the introduction, variables of conjunctive types are useless, hence all variables have prime types. This way, there is no term $\lambda x^{\tau\wedge\tau}.x$, but a term $\lambda y^\tau.\lambda z^\tau.y\times z$ which is equivalent to $(\lambda y^\tau.\lambda z^\tau.y)\times(\lambda y^\tau.\lambda z^\tau.z)$.

The size of a term is not invariant through the equivalence $\rightleftarrows$. Hence, we introduce a measure $M(\cdot)$, which is given in Table 3.

Table 3 Measure on terms.

$P(x)$	$= 0$	$M(x)$	$= 1$
$P(\lambda x^A.r)$	$= P(r)$	$M(\lambda x^A.r)$	$= 1 + M(r) + P(r)$
$P(rs)$	$= P(r)$	$M(rs)$	$= M(r) + M(s) + P(r)M(s)$
$P(r \times s)$	$= 1 + P(r) + P(s)$	$M(r \times s)$	$= M(r) + M(s)$
$P(\pi_A(r))$	$= P(r)$	$M(\pi_A(r))$	$= 1 + M(r) + P(r)$

▶ **Lemma 3.2.** *If* $r \rightleftarrows s$ *then* $P(r) = P(s)$.

Proof. We check the case of each rule of Table 2, and then conclude by structural induction to handle the contextual closure.

- (COMM): $P(r \times s) = 1 + P(r) + P(s) = P(s \times r)$.
- (ASSO): $P((r \times s) \times t) = 2 + P(r) + P(s) + P(t) = P(r \times (s \times t))$.
- (DIST$_\lambda$): $P(\lambda x^A.(r \times s)) = 1 + P(r) + P(s) = P(\lambda x^A.r \times \lambda x^A.s)$.
- (DIST$_{\mathsf{app}}$): $P((r \times s)t) = 1 + P(r) + P(s) = P(rt \times st)$.
- (CURRY): $P((rs)t) = P(r) = P(r(s \times t))$. ◀

▶ **Lemma 3.3.** *If* $r \rightleftarrows s$ *then* $M(r) = M(s)$.

Proof. We check the case of each rule of Table 2, and then conclude by structural induction to handle the contextual closure.

- (COMM): $M(r \times s) = M(r) + M(s) = M(s \times r)$.
- (ASSO): $M((r \times s) \times t) = M(r) + M(s) + M(t) = M(r \times (s \times t))$.
- (DIST$_\lambda$): $M(\lambda x^A.(r \times s)) = 2 + M(r) + M(s) + P(r) + P(s) = M(\lambda x^A.r \times \lambda x^A.s)$
- (DIST$_{\mathsf{app}}$): $M((r \times s)t) = M(r) + M(s) + 2M(t) + P(r)M(t) + P(s)M(t) = M(rt \times st)$
- (CURRY): $M((rs)t) = M(r) + M(s) + P(r)M(s) + M(t) + P(r)M(t) = M(r(s \times t))$ ◀

▶ **Lemma 3.4.** $M(\lambda x^A.r) > M(r)$*,* $M(rs) > M(r)$*,* $M(rs) > M(s)$*,* $M(r \times s) > M(r)$*,* $M(r \times s) > M(s)$*, and* $M(\pi_A(r)) > M(r)$*.*

Proof. By induction on r, $M(r) \geq 1$. We conclude with a case inspection. ◀

We use the measure to prove that the equivalence class of a term is a finite set.

▶ **Lemma 3.5.** *For any term* r*, the set* $\{s \mid s \rightleftarrows^* r\}$ *is finite (modulo* α*-equivalence).*

Proof. Since $\{s \mid s \rightleftarrows^* r\} \subseteq \{s \mid FV(s) = FV(r) \text{ and } M(s) = M(r)\} \subseteq \{s \mid FV(s) \subseteq FV(r) \text{ and } M(s) \leq M(r)\}$, where $FV(t)$ is the set of free variables of t, all we need to prove is that for all natural numbers n, for all finite sets of variables F, the set $H(n, F) = \{s \mid FV(s) \subseteq F \text{ and } M(s) \leq n\}$ is finite.

By induction on n. For $n = 1$ the set $\{s \mid FV(s) \subseteq F \text{ and } M(s) \leq 1\}$ contains only the variables of F. Assume the property holds for n, then, by the Lemma 3.4 the set $H(n+1, F)$ is a subset of the finite set containing the variables of F, the abstractions $(\lambda x^A.r)$ for r in $H(n, F \cup \{x\})$, the applications (rs) for r and s in $H(n, F)$, the products $r \times s$ for r and s in $H(n, F)$, the projections $\pi_A(r)$ for r in $H(n, F)$. ◀

▶ **Definition 3.6.** *The reduction relation* $\hookrightarrow$ *is the smallest contextually closed relation defined by the rules given in Table 4. We write* $\hookrightarrow^*$ *for the transitive and reflexive closure of* $\hookrightarrow$*.*

Table 4 Reduction relation.

If $s : A$, $(\lambda x^A.r)s \hookrightarrow r[s/x]$	(β)	If $r : A$, $\pi_A(r \times s) \hookrightarrow r$	(π)

▶ **Definition 3.7.** *We write $\rightsquigarrow$ for the relation $\hookrightarrow$ modulo $\rightleftarrows^*$ (i.e. $r \rightsquigarrow s$ iff $r \rightleftarrows^* r' \hookrightarrow s' \rightleftarrows^* s$), and $\rightsquigarrow^*$ for its transitive and reflexive closure.*

Remark that, by Lemma 3.5, a term has a finite number of reducts in one step and these reducts can be computed.

3.3 Examples

▶ **Example 3.8.** Let $r : A$ and $s : B$. Then $(\lambda x^A.\lambda y^B.x)(r \times s) : A$ and

$$(\lambda x^A.\lambda y^B.x)(r \times s) \rightleftarrows (\lambda x^A.\lambda y^B.x)rs \hookrightarrow^* r$$

However, if $A \equiv B$, it is also possible to reduce in the following way

$$(\lambda x^A.\lambda y^A.x)(r \times s) \rightleftarrows (\lambda x^A.\lambda y^A.x)(s \times r) \rightleftarrows (\lambda x^A.\lambda y^A.x)sr \hookrightarrow^* s$$

Hence, the usual encoding of the projector also behaves non-deterministically.

▶ **Example 3.9.** Let $s : A$ and $t : B$, and let $\mathbf{TF} = \lambda x^A.\lambda y^B.(x \times y)$.

Then $\mathbf{TF} : A \Rightarrow B \Rightarrow (A \wedge B) \equiv ((A \wedge B) \Rightarrow A) \wedge ((A \wedge B) \Rightarrow B)$. Therefore, $\pi_{(A\wedge B)\Rightarrow A}(\mathbf{TF}) : (A \wedge B) \Rightarrow A$. Hence, $\pi_{(A\wedge B)\Rightarrow A}(\mathbf{TF})(s \times t) : A$.

This term reduces as follows:

$$\begin{aligned}
\pi_{(A\wedge B)\Rightarrow A}(\mathbf{TF})(s \times t) &\rightleftarrows \pi_{(A\wedge B)\Rightarrow A}(\mathbf{TF})st \\
&\rightleftarrows \pi_{(A\wedge B)\Rightarrow A}(\lambda x^A.(\lambda y^B.x) \times (\lambda y^B.y))st \\
&\rightleftarrows \pi_{(A\wedge B)\Rightarrow A}((\lambda x^A.\lambda y^B.x) \times (\lambda x^A.\lambda y^B.y))st \\
&\hookrightarrow (\lambda x^A.\lambda y^B.x)st \\
&\hookrightarrow (\lambda y^B.s)t \hookrightarrow s
\end{aligned}$$

▶ **Example 3.10.** Let $\mathbf{T} = \lambda x^A.\lambda y^B.x$ and $\mathbf{F} = \lambda x^A.\lambda y^B.y$. The term $\mathbf{T} \times \mathbf{F} \times \mathbf{TF}$ has type $((A \wedge B) \Rightarrow (A \wedge B)) \wedge ((A \wedge B) \Rightarrow (A \wedge B))$.

Hence, $\pi_{(A\wedge B)\Rightarrow(A\wedge B)}(\mathbf{T} \times \mathbf{F} \times \mathbf{TF})$ is well typed and reduces non-deterministically either to $\mathbf{T} \times \mathbf{F}$ or to $\mathbf{TF}$. Moreover, as $\mathbf{T} \times \mathbf{F}$ and $\mathbf{TF}$ are equivalent, the non-deterministic choice does not play any role in this particular case. We will come back to the encoding of booleans in System I on Section 7.

4 Subject Reduction

The set of types assigned to a term is preserved under $\rightleftarrows$ and $\hookrightarrow$. Before proving this property, we prove the unicity of types (Lemma 4.1), the generation lemma (Lemma 4.2), and the substitution lemma (Lemma 4.3). We only state the lemmas in this section. The detailed proofs can be found in Appendix A.

The following lemma states that a term can be typed only by equivalent types.

▶ **Lemma 4.1** (Unicity). *If $r : A$ and $r : B$, then $A \equiv B$.*

Proof.
- If the last rule of the derivation of $r : A$ is $(\equiv)$, then we have a shorter derivation of $r : C$ with $C \equiv A$, and, by the induction hypothesis, $C \equiv B$, hence $A \equiv B$.
- If the last rule of the derivation of $r : B$ is $(\equiv)$ we proceed in the same way.
- All the remaining cases are syntax directed. ◀

▶ **Lemma 4.2** (Generation).
1. *If $x \in \mathcal{V}_A$ and $x : B$, then $A \equiv B$.*
2. *If $\lambda x^A.r : B$, then $B \equiv A \Rightarrow C$ and $r : C$.*
3. *If $rs : B$, then $r : A \Rightarrow B$ and $s : A$.*
4. *If $r \times s : A$, then $A \equiv B \wedge C$ with $r : B$ and $s : C$.*
5. *If $\pi_A(r) : B$, then $A \equiv B$ and $r : B \wedge C$.*

Proof. Each statement is proved by induction on the typing derivation. For the statement 1, we have $x \in \mathcal{V}_A$ and $x : B$. The only way to type this term is either by the rule (ax) or $(\equiv)$.
- In the first case, $A = B$, hence $A \equiv B$.
- In the second case, there exists B' such that $x : B'$ has a shorter derivation, and $B \equiv B'$. By the induction hypothesis $A \equiv B' \equiv B$.

For the statement 2, we have $\lambda x^A.r : B$. The only way to type this term is either by rule $(\Rightarrow_i)$, $(\equiv)$.
- In the first case, we have $B = A \Rightarrow C$ for some, C and $r : C$.
- In the second, there exists B' such that $\lambda x^A.r : B'$ has a shorter derivation, and $B \equiv B'$. By the induction hypothesis, $B' \equiv A \Rightarrow C$ and $r : C$. Thus, $B \equiv B' \equiv A \Rightarrow C$.

The three other statements are similar. ◀

▶ **Lemma 4.3** (Substitution). *If $r : A$, $s : B$, and $x \in \mathcal{V}_B$, then $r[s/x] : A$.*

Proof. By structural induction on r (cf. Appendix A). ◀

▶ **Theorem 4.4** (Subject reduction). *If $r : A$ and $r \hookrightarrow s$ or $r \rightleftarrows s$ then $s : A$.*

Proof. By induction on the rewrite relation (cf. Appendix A). ◀

5 Strong Normalisation

In this section we prove the strong normalisation of reduction $\rightsquigarrow$: every reduction sequence fired from a typed term eventually terminates. The set of strongly normalising terms with respect to reduction $\rightsquigarrow$ is written SN. The size of the longest reduction issued from t is written $|t|$ (recall that each term has a finite number of reducts).

To prove that every term is in SN, we associate, as usual, a set $[\![A]\!]$ of strongly normalising terms to each type A. A term $r : A$ is said to be reducible when $r \in [\![A]\!]$. We then prove an adequacy theorem stating that every well typed term is reducible.

In simply typed lambda calculus we can either define $[\![A_1 \Rightarrow A_2 \Rightarrow \cdots \Rightarrow A_n \Rightarrow \tau]\!]$ as the set of terms r such that for all $s \in [\![A_1]\!]$, $rs \in [\![A_2 \Rightarrow \cdots \Rightarrow A_n \Rightarrow \tau]\!]$ or, equivalently, as the set of terms r such that for all $s_i \in [\![A_i]\!]$, $rs_1 \dots s_n \in [\![\tau]\!] = \mathsf{SN}$. To prove that a term of the form $\lambda x^A.t$ is reducible, we need to use the so-called CR3 property [22], in the first case, and the property that a term whose all one-step reducts are in SN is in SN, in the second. In System I, an introduction can be equivalent to an elimination e.g. $rt \times st \rightleftarrows (r \times s)t$, hence, we cannot define a notion of neutral term and have an equivalent to the CR3 property. Therefore, we use the second definition.

Before we prove the normalisation of System I, we first reformulate the proof of strong normalisation of simply typed lambda-calculus along these lines.

5.1 Normalisation of simply typed lambda calculus

▶ **Definition 5.1** (Elimination context). *Consider an extension of simply typed lambda calculus where we introduce an extra symbol* $[]^A$*, called hole of type* A*.*

An elimination context with a hole $[]^{B_1 \Rightarrow \cdots \Rightarrow B_n \Rightarrow \tau}$ *is a term* $K^\tau_{B_1 \Rightarrow \cdots \Rightarrow B_n \Rightarrow \tau}$ *of type* τ *of the form* $[]^{B_1 \Rightarrow \cdots \Rightarrow B_n \Rightarrow \tau} r_1 \dots r_n$*. We write* $K^\tau_A[t]$*, for the term* $K^\tau_A[t/[]^A] = tr_1 \dots r_n$*.*

▶ **Definition 5.2** (Terms occurring in an elimination context). $\mathcal{T}([]^A r_1 \dots r_n) = \{r_1, \dots, r_n\}$*. Note that the types of the elements of* $\mathcal{T}([]^A r_1 \dots r_n)$ *are smaller than* A*, and that if* $r_1, \dots, r_n \in \mathsf{SN}$*, then* $[]^A r_1 \dots r_n \in \mathsf{SN}$*.*

▶ **Definition 5.3** (Reducibility). *The set* $[\![A]\!]$ *of reducible terms of type* A *is defined by structural induction on* A *as the set of terms* $t : A$ *such that for any elimination context* K^τ_A *such that the terms in* $\mathcal{T}(K^\tau_A)$ *are all reducible, we have* $K^\tau_A[t] \in \mathsf{SN}$*.*

▶ **Definition 5.4** (Reducible elimination context). *An elimination context* K^B_A *is reducible, if all the terms in* $\mathcal{T}(K^B_A)$ *are reducible.*

▶ **Lemma 5.5.** *For all* A*,* $[\![A]\!] \subseteq \mathsf{SN}$ *and all the variables of type* A *are in* $[\![A]\!]$*.*

Proof. By induction on A. ◀

▶ **Lemma 5.6** (Adequacy of application). *If* $r \in [\![A \Rightarrow B]\!]$ *and* $s \in [\![A]\!]$*, then* $rs \in [\![B]\!]$*.*

Proof. Let K^τ_B be a reducible elimination context. We need to prove that $K^\tau_B[rs] \in \mathsf{SN}$. As $s \in [\![A]\!]$, the elimination context $K'^\tau_{A \Rightarrow B} = K^\tau_B[[]^{A \Rightarrow B} s]$ is reducible, and since $r \in [\![A \Rightarrow B]\!]$, we have $K^\tau_B[rs] = K'^\tau_{A \Rightarrow B}[r] \in \mathsf{SN}$. ◀

▶ **Lemma 5.7** (Adequacy of abstraction). *If for all* $t \in [\![A]\!]$*,* $r[t/x] \in [\![B]\!]$*, then* $\lambda x^A.r \in [\![A \Rightarrow B]\!]$*.*

Proof. We need to prove that for every reducible elimination context $K^\tau_{A \Rightarrow B}$, $K^\tau_{A \Rightarrow B}[\lambda x^A.r] \in \mathsf{SN}$, that is that all its one step reducts are in SN. By Lemma 5.5, $x \in [\![A]\!]$, so $r \in [\![B]\!] \subseteq \mathsf{SN}$. Then, we proceed by induction on $|r| + |K^\tau_{A \Rightarrow B}|$. ◀

▶ **Definition 5.8** (Adequate substitution). *A substitution* σ *is adequate if for all* $x : A$*, we have* $\sigma(x) \in [\![A]\!]$*.*

▶ **Theorem 5.9** (Adequacy). *If* $r : A$*, the for all* σ *adequate, we have* $\sigma r \in [\![A]\!]$*.*

Proof. By induction on r. ◀

▶ **Theorem 5.10** (Strong normalisation). *If* $r : A$*, then* $r \in \mathsf{SN}$*.*

Proof. By Lemma 5.5, the idendity substitution is adequate. Thus, by Theorem 5.9 and Lemma 5.5, $r \in [\![A]\!] \subseteq \mathsf{SN}$. ◀

5.2 Reduction of a product

When simply-typed lambda-calculus is extended with pairs, proving that if $r_1 \in \mathsf{SN}$ and $r_2 \in \mathsf{SN}$ then $r_1 \times r_2 \in \mathsf{SN}$ is easy. However, in System I this property (Lemma 5.13) is harder to prove, as it requires a characterisation of the terms equivalent to the product $r_1 \times r_2$ (Lemma 5.11) and of all the reducts of this term (Lemma 5.12).

In Lemma 5.11, we characterise the terms equivalent to a product.

▶ **Lemma 5.11.** *If $r \times s \rightleftarrows^* t$ then either*

1. $t = u \times v$ *where either*
   a. $u \rightleftarrows^* t_{11} \times t_{21}$ *and* $v \rightleftarrows^* t_{12} \times t_{22}$ *with* $r \rightleftarrows^* t_{11} \times t_{12}$ *and* $s \rightleftarrows^* t_{21} \times t_{22}$*, or*
   b. $v \rightleftarrows^* w \times s$ *with* $r \rightleftarrows^* u \times w$*, or any of the three symmetric cases, or*
   c. $r \rightleftarrows^* u$ *and* $s \rightleftarrows^* v$*, or the symmetric case.*
2. $t = \lambda x^A.a$ *and* $a \rightleftarrows^* a_1 \times a_2$ *with* $r \rightleftarrows^* \lambda x^A.a_1$ *and* $s \rightleftarrows^* \lambda x^A.a_2$*.*
3. $t = av$ *and* $a \rightleftarrows^* a_1 \times a_2$*, with* $r \rightleftarrows^* a_1 v$ *and* $s \rightleftarrows^* a_2 v$*.*

Proof. By a double induction, first on $M(t)$ and then on the length of the relation $\rightleftarrows^*$ (cf. Appendix B.1). ◀

In Lemma 5.12, we characterise the reducts of a product.

▶ **Lemma 5.12.** *If $r_1 \times r_2 \rightleftarrows^* s \hookrightarrow t$, there exists u_1, u_2 such that $t \rightleftarrows^* u_1 \times u_2$ and either ($r_1 \rightsquigarrow u_1$ and $r_2 \rightsquigarrow u_2$), or ($r_1 \rightsquigarrow u_1$ and $r_2 \rightleftarrows^* u_2$), or ($r_1 \rightleftarrows^* u_1$ and $r_2 \rightsquigarrow u_2$).*

Proof. By induction on $M(r_1 \times r_2)$. ◀

▶ **Lemma 5.13.** *If $r_1 \in \mathsf{SN}$ and $r_2 \in \mathsf{SN}$, then $r_1 \times r_2 \in \mathsf{SN}$.*

Proof. By Lemma 5.12, from a reduction sequence starting from $r_1 \times r_2$ we can extract one starting from r_1, or r_2 or both. Hence, this reduction sequence is finite. ◀

5.3 Reduction of a term of a conjunctive type

The next lemma takes advantage of the fact that all the variables have prime types to prove that all terms of conjunctive type, even open ones, reduce to a product. For instance, instead of the term $\lambda x^{\tau \wedge \tau}.x$, of type $((\tau \wedge \tau) \Rightarrow \tau) \wedge ((\tau \wedge \tau) \Rightarrow \tau)$, we must write $\lambda y^\tau.\lambda z^\tau.y \times z$, which is equivalent to $(\lambda y^\tau.\lambda z^\tau.y) \times (\lambda y^\tau.\lambda z^\tau.z)$.

▶ **Lemma 5.14.** *If $r : \bigwedge_{i=1}^n A_i$, then $r \rightsquigarrow^* \prod_{i=1}^n r_i$ where $r_i : A_i$.*

Proof. By induction on r.

- $r = x$, then it has a prime type, so take $r_1 = r$.
- Let $r = \lambda x^C.s$. Then, by Lemma 4.2, $s : D$ with $C \Rightarrow D \equiv \bigwedge_i A_i$. So, by Corollary 2.12, $D \equiv \bigwedge_i D_i$, and so, by the induction hypothesis, $s \rightsquigarrow^* \prod_i s_i$. Therefore, $\lambda x^C.s \rightsquigarrow^* \prod_i \lambda x^C.s_i$.
- Let $r = st$. Then, by Lemma 4.2, $s : C \Rightarrow \bigwedge_i A_i$, so $s : \bigwedge_i (C \Rightarrow A_i)$. Therefore, by the induction hypothesis, $s \rightsquigarrow^* \prod_i s_i$, and so $st \rightsquigarrow^* \prod_i s_i t$.
- Let $r = s \times t$. Then, by Lemma 4.2, $s : B$ and $t : C$, with $B \wedge C \equiv \bigwedge_i A_i$. By Lemma 2.15, there exists a partition $E \uplus F \uplus G$ of $\{1, \dots, n\}$ such that $A_i \equiv B_i \wedge C_i$, when $i \in E$; $A_i \equiv B_i$, when $i \in F$; $A_i \equiv C_i$, when $i \in G$; $B \equiv \bigwedge_{i \in E \uplus F} B_i$; and $C \equiv \bigwedge_{i \in E \uplus G} C_i$. By the induction hypothesis, $s \rightsquigarrow^* \prod_{i \in E \uplus F} s_i$ and $t \rightsquigarrow^* \prod_{i \in E \uplus G} t_i$. If $i \in E$, we let $r_i = s_i \times t_i$, if $i \in F$, we let $r_i = s_i$, if $i \in G$, we let $r_i = t_i$. We have $r = s \times t \rightsquigarrow^* \prod_{i=1}^n r_i$.
- Let $r = \pi_{\bigwedge_i A_i}(s)$. Then, by Lemma 4.2, $s : \bigwedge_i A_i \wedge B$, and hence, by the induction hypothesis, $s \rightsquigarrow^* \prod_i s_i \times t$ where $s_i : A_i$ and $t : B$, hence $r \rightsquigarrow \prod_i s_i$. ◀

▶ **Corollary 5.15.** *If $r : A \wedge B$, then $r \rightsquigarrow^* r_1 \times r_2$ where $r_1 : A$ and $r_2 : B$.*

Proof. Let $\mathsf{PF}(A) = [A_i]_i$, $\mathsf{PF}(B) = [B_j]_j$, by Lemma 2.6, $A \wedge B \equiv \bigwedge_i A_i \wedge \bigwedge_j B_j$. Then, by Lemma 5.14, $r \rightsquigarrow^* \prod_i r_{1i} \times \prod_j r_{2j}$. Take $r_1 = \prod_i r_{1i}$ and $r_2 = \prod_j r_{2j}$. ◀

5.4 Reducibility

▶ **Definition 5.16** (Elimination context). *Consider an extension of the language where we introduce an extra symbol* $[]^A$*, called hole of type* A*. We define the set of elimination contexts with a hole* $[]^A$ *as the smallest set such that:*

- $[]^A$ *is an elimination context of type* A*,*
- *if* $K_A^{B\Rightarrow C}$ *is an elimination context of type* $B \Rightarrow C$ *with a hole of type* A*, and* $r : B$ *then* $K_A^{B\Rightarrow C} r$ *is an elimination context of type* C *with a hole of type* A*,*
- *and if* $K_A^{B\wedge C}$ *is an elimination context of type* $B \wedge C$ *with a hole of type* A*, then* $\pi_B(K_A^{B\wedge C})$ *is an elimination context of type* B *with a hole of type* A*.*

We write $K_A^B[t]$ *for* $K_A^B[t/[]^A]$*, where* $[]^A$ *is the hole of* K_A^B*. In particular,* t *may be an elimination context.*

▶ **Example 5.17.** Let $K_\tau^\tau = []^\tau$ and $K'^\tau_{\tau\Rightarrow(\tau\wedge\tau)} = K_\tau^\tau[\pi_\tau([]^{\tau\Rightarrow(\tau\wedge\tau)} x)]$. Then $K'^\tau_{\tau\Rightarrow(\tau\wedge\tau)} = \pi_\tau([]^{\tau\Rightarrow(\tau\wedge\tau)} x)$, and $K'^\tau_{\tau\Rightarrow(\tau\wedge\tau)}[\lambda y^\tau.y \times y] = \pi_\tau((\lambda y^\tau.y \times y)x)$.

▶ **Definition 5.18** (Terms occurring in an elimination context). *Let* K_A^B *be an elimination context. The multiset of terms occurring in* K_A^B *is defined as*

$$\mathcal{T}([]^A) = \emptyset; \qquad \mathcal{T}(K_A^{B\Rightarrow C} r) = \mathcal{T}(K_A^{B\Rightarrow C}) \uplus \{r\}; \qquad \mathcal{T}(\pi_B(K_A^{B\wedge C})) = \mathcal{T}(K_A^{B\wedge C})$$

We write $|K_A^B|$ *for* $\sum_{i=1}^n |r_i|$ *where* $[r_1, \dots, r_n] = \mathcal{T}(K_A^B)$*.*

▶ **Example 5.19.** $\mathcal{T}([]^A rs) = [r, s]$ and $\mathcal{T}([]^A(r \times s)) = [r \times s]$. Remark that $K_A^B[t] \rightleftarrows^* K'^B_A[t]$ does not imply $\mathcal{T}(K_A^B) \sim \mathcal{T}(K'^B_A)$.

Remark that if K_A^B is a context, $m(B) \leq m(A)$ and hence if a term in $\mathcal{T}(K_A^B)$ has type C, then $m(C) < m(A)$.

▶ **Lemma 5.20.** *Let* K_A^τ *be an elimination context such that* $\mathcal{T}(K_A^\tau) \subseteq \mathsf{SN}$*, let* $\mathsf{PF}(A) = [B_1, \dots, B_n]$*, and let* $x_i \in \mathcal{V}_{B_i}$*. Then* $K_A^\tau[x_1 \times \dots \times x_n] \in \mathsf{SN}$*.*

Proof. By induction on the number of projections in K_A^τ.

- If K_A^τ does not contain any projection, then it has the form $[]^A\ r_1\ \dots\ r_m$. Let C_i be the type of r_i, we have $A \equiv C_1 \Rightarrow \dots \Rightarrow C_n \Rightarrow \tau$, thus A is prime, $n = 1$, and we need to prove that $x_1\ r_1\ \dots\ r_m$ is in SN which is a consequence of the fact that r_1, ..., r_n are in SN.
- Otherwise, $K_A^\tau = K'^\tau_B[\pi_B([]^A r_1 \dots r_m)]$, and $K_A^\tau[x_1 \times \dots \times x_n] = K'^\tau_B[\pi_B((x_1 \times \dots \times x_n) r_1 \dots r_m)] \rightleftarrows^* K'^\tau_B[\pi_B(x_1 r_1 \dots r_m \times \dots \times x_n r_1 \dots r_m)]$. We prove that this term is in SN by showing, more generally, that if s_i are reducts of $x_i r_1 \dots r_m$, then $K'^\tau_B[\pi_B(s_1 \times \dots \times s_n)] \in \mathsf{SN}$. To do so, we show, by induction on $|K'^\tau_B| + |s_1 \times \dots \times s_n|$, that all the one step reducts of this term are in SN.
  - If the reduction takes place in one of the terms in $\mathcal{T}(K'^\tau_B)$ or in one of the s_i, we apply the induction hypothesis.
  - Otherwise, the reduction is a (π) reduction of $\pi_B(s_1 \times \dots \times s_n)$ yielding, without lost of generality, a term of the form $K'^\tau_B[s_1 \times \dots \times s_q]$. This term is a reduct of $K'^\tau_B[(x_1 \times \dots \times x_q) r_1 \dots r_m]$. As the context $K'^\tau_B[([]^C\ r_1 \dots r_m)]$ contains one projection less than K_A^τ this term is in SN. Hence so does its reduct $K'^\tau_B[s_1 \times \dots \times s_q]$. ◀

▶ **Definition 5.21** (Reducibility). *The set* $[\![A]\!]$ *of reducible terms of type* A *is defined by induction on* $m(A)$ *as the set of terms* $t : A$ *such that for any elimination context* K_A^τ *such that the terms of* $\mathcal{T}(K_A^\tau)$ *are all reducible, we have* $K_A^\tau[t] \in \mathsf{SN}$*.*

▶ **Definition 5.22** (Reducible elimination context). *An elimination context K_A^B is reducible, if all the terms in $\mathcal{T}(K_A^B)$ are reducible.*

From now on we consider all the elimination contexts to be reducible.

The following lemma is a trivial consequence of the definition of reducibility.

▶ **Lemma 5.23.** *If $A \equiv B$, then $[\![A]\!] = [\![B]\!]$.*

▶ **Lemma 5.24.** *For all A, $[\![A]\!] \subseteq \mathsf{SN}$ and $[\![A]\!] \neq \emptyset$.*

Proof. By induction on $m(A)$. By the induction hypothesis, for all the B such that $m(B) < m(A)$, $[\![B]\!] \neq \emptyset$. Thus, there exists an elimination context K_A^τ. Hence, if $r \in [\![A]\!]$, $K_A^\tau[r] \in \mathsf{SN}$, hence $r \in \mathsf{SN}$.

We then prove that if $\mathsf{PF}(A) = [B_1, \dots, B_n]$ and $x_i \in \mathcal{V}_{B_i}$ then $\prod_i x_i \in [\![A]\!]$. By the induction hypothesis, $\mathcal{T}(K_A^\tau) \subseteq \mathsf{SN}$, hence, by Lemma 5.20, $K_A^\tau[x_1 \times \dots \times x_n] \in \mathsf{SN}$. ◀

5.5 Adequacy

We finally prove the adequacy theorem (Theorem 5.30) showing that every typed term is reducible, and the strong normalisation theorem (Theorem 5.31) as a consequence of it.

▶ **Lemma 5.25** (Adequacy of projection). *If $r \in [\![A \wedge B]\!]$, then $\pi_A(r) \in [\![A]\!]$.*

Proof. We need to prove that $K_A^\tau[\pi_A(r)] \in \mathsf{SN}$. Take $K'^\tau_{A\wedge B} = K_A^\tau[\pi_A[]^{A\wedge B}]$ and since $r \in [\![A \wedge B]\!]$, we have $K_A^\tau[\pi_A(r)] = K'^\tau_{A\wedge B}[r] \in \mathsf{SN}$. ◀

▶ **Lemma 5.26** (Adequacy of application). *If $r \in [\![A \Rightarrow B]\!]$, and $s \in [\![A]\!]$, then $rs \in [\![B]\!]$.*

Proof. We need to prove that $K_B^\tau[rs] \in \mathsf{SN}$. Take $K'^\tau_{A\Rightarrow B} = K_B^\tau[[]^{A\Rightarrow B}s]$ and since $r \in [\![A \Rightarrow B]\!]$, we have $K_B^\tau[rs] = K'^\tau_{A\Rightarrow B}[r] \in \mathsf{SN}$. ◀

▶ **Lemma 5.27** (Adequacy of product). *If $r \in [\![A]\!]$ and $s \in [\![B]\!]$, then $r \times s \in [\![A \wedge B]\!]$.*

Proof. We need to prove that $K^\tau_{A\wedge B}[r \times s] \in \mathsf{SN}$. We proceed by induction on the number of projections in $K^\tau_{A\Rightarrow B}$. Since the hole of $K^\tau_{A\wedge B}$ has type $A \wedge B$, and $K^\tau_{A\wedge B}[t]$ has type τ for any $t : A$, we can assume, without lost of generality, that the context $K^\tau_{A\wedge B}$ has the form $K'^\tau_C[\pi_C([]^{A\wedge B}t_1 \dots t_n)]$. We prove that all $K'^\tau_C[\pi_C(rt_1 \dots t_n \times st_1 \dots t_n)] \in \mathsf{SN}$ by showing, more generally, that if r' and s' are two reducts of $rt_1 \dots t_n$ and $st_1 \dots t_n$, then $K'^\tau_C[\pi_C(r' \times s')] \in \mathsf{SN}$. For this, we show that all its one step reducts are in SN, by induction on $|K'^\tau_C| + |r'| + |s'|$. Full details are given in Appendix B.2. ◀

▶ **Lemma 5.28** (Adequacy of abstraction). *If for all $t \in [\![A]\!]$, $r[t/x] \in [\![B]\!]$, then $\lambda x^A.r \in [\![A \Rightarrow B]\!]$.*

Proof. By induction on $M(r)$. In the case $r \not\rightleftarrows^* r_1 \times r_2$, we need to prove that for any elimination context $K^\tau_{A\Rightarrow B}$, we have $K^\tau_{A\Rightarrow B}[\lambda x^A.r] \in \mathsf{SN}$, and we do so by a second induction on $|K^\tau_{A\Rightarrow B}| + |r|$ to show that all the one step reducts of $K^\tau_{A\Rightarrow B}[\lambda x^A.r]$ are in SN. Full details are given in Appendix B.2. ◀

▶ **Definition 5.29** (Adequate substitution). *A substitution σ is adequate if for all $x \in \mathcal{V}_A$, we have $\sigma(x) \in [\![A]\!]$.*

▶ **Theorem 5.30** (Adequacy). *If $r : A$, then for all σ adequate, we have $\sigma r \in [\![A]\!]$.*

Proof. By induction on r. Full details are given in Appendix B.2. ◀

▶ **Theorem 5.31** (Strong normalisation). *If $r : A$, then $r \in \mathsf{SN}$.*

Proof. By Lemma 5.24, the identity substitution is adequate. Thus, by Theorem 5.30 and Lemma 5.24, $r \in [\![A]\!] \subseteq \mathsf{SN}$. ◀

6 Consistency

▶ **Lemma 6.1.** *For any term $r : A$ there exists an elimination context K^A_B and a term $s : B$, which is not an elimination, such that $r = K^A_B[s]$.*

Proof. We proceed by structural induction on r.

- If r is a variable, an abstraction, or a product, we take $s = r$ and $K^A_A = []^A$.
- If r is an application $r_1 r_2$, by the induction hypothesis, $r_1 = K^{C\Rightarrow B}_A[s]$, we take $K'^B_A = K^{C\Rightarrow B}_A r_2$.
- If r is a projection $\pi_A(r')$, by the induction hypothesis, $r' = K^{A\wedge C}_B[s]$, we take $K'^A_B = \pi_A(K^{A\wedge C}_B)$. ◀

▶ **Corollary 6.2.** *There is no closed normal term of type τ.*

Proof. Let $r : \tau$ be a closed normal term. By Lemma 6.1, any $r = K^\tau_A[s]$, where s is not an elimination. Since the term is closed, s is not a variable. Thus it is either and abstraction or a product.

- If A is prime, then, K^τ_A cannot contain a projection, so by rule (CURRY) we have $K^\tau_A \rightleftarrows^* []^A t$, with $t : B$, and s has the form $\lambda x^C.s'$ with $s' : D \Rightarrow \tau$. We have $B \equiv C \wedge D$. By Corollary 5.15, and since t is normal, $t \rightleftarrows^* t_1 \times t_2$ where $t_1 : C$ and $t_2 : D$, so $K^\tau_A \rightleftarrows^* []^A t_1 t_2$, hence, $r = K^\tau_A[\lambda x^C.s'] \rightleftarrows^* (\lambda x^C.s')t_1 t_2$ is not normal.
- Otherwise, $K^\tau_A = K'^\tau_B[\pi_B([]^A t_1 \dots t_n)]$, with $[]^A t_1 \dots t_n : B \wedge C$. Then, by Corollary 5.15, and since $st_1 \dots t_n$ is normal, $st_1 \dots t_n \rightleftarrows^* s_1 \times s_2$, thus $r = K^\tau_A[s] \rightleftarrows^* K'^\tau_B[\pi_B(s_1 \times s_2)]$, which is not normal. ◀

7 Computing with System I

Because the symbol $\times$ is associative and commutative, System I does not contain the usual notion of pairs. However, it is possible to encode a deterministic projection, even if we have more than one term of the same type. An example, although there are various possibilities, is to encode the pairs $\langle r, s\rangle : A \times A$ as $\lambda x^{\mathbb{1}}.r \times \lambda x^{\mathbb{2}}.s : \mathbb{1} \Rightarrow A \wedge \mathbb{2} \Rightarrow A$ and the projection $\pi_1\langle r, s\rangle$ as $\pi_{\mathbb{1}\Rightarrow A}(\lambda x^{\mathbb{1}}.r \times \lambda x^{\mathbb{2}}.s)y^{\mathbb{1}}$ (similarly for π_2), where types $\mathbb{1}$ and $\mathbb{2}$ are any two different types. This example uses free variables, but it is easy to close it, e.g. use $\lambda y.y$ instead of $y^{\mathbb{1}}$ in the second line. Moreover, this technique is not limited to pairs. Due to the associativity of $\times$, the encoding can be easily extended to lists.

Example 3.10 on booleans overlooks an interesting fact: If $A \equiv B$, then both $\mathbf{T}$ and $\mathbf{F}$ behave as a non-deterministic projector. Indeed, $\mathbf{T}rs \hookrightarrow^* r$, but also $(\lambda x^A.\lambda y^A.x)rs \rightleftarrows (\lambda x^A.\lambda y^A.x)(r \times s) \rightleftarrows (\lambda x^A.\lambda y^A.x)(s \times r) \rightleftarrows (\lambda x^A.\lambda y^A.x)sr \hookrightarrow^* s$. Similarly, $\mathbf{F}rs \hookrightarrow^* s$ and also $\mathbf{F}rs \rightsquigarrow^* r$. Hence, $A \Rightarrow A \Rightarrow A$ is not suitable to encode the type Bool. The type $A \Rightarrow A \Rightarrow A$ has only one term in the underlying equational theory.

Fortunately, there are ways to construct types with more than one term. First, let us define the following notation. For any t, we write $[t]^{\tau\Rightarrow\tau}$, the *canon* of t, that is, the term $\lambda z^{\tau\Rightarrow\tau}.t$, where $z^{\tau\Rightarrow\tau}$ is a fresh variable not appearing in t. Also, for any term t of type $(\tau \Rightarrow \tau) \Rightarrow A$, we write $\{t\}^{\tau\Rightarrow\tau}$, the *cocanon*, which is the inverse operation, that is, $\{[t]^{\tau\Rightarrow\tau}\}^{\tau\Rightarrow\tau} \hookrightarrow t$ for any t of type A. For the cocanon it suffices to take $\{t\}^{\tau\Rightarrow\tau} = t(\lambda x^\tau.x)$.

Therefore, the type $((\tau \Rightarrow \tau) \Rightarrow A) \Rightarrow A \Rightarrow A$ has the following two different terms: $\mathbf{tt} := \lambda x^A.\lambda y^{(\tau\Rightarrow\tau)\Rightarrow A}.x$ and $\mathbf{ff} := \lambda x^{(\tau\Rightarrow\tau)\Rightarrow A}.\lambda y^A.\{x\}^{\tau\Rightarrow\tau}$. Hence, it is possible to encode an if-then-else conditional expression as $\mathsf{If}\ \mathsf{c}\ \mathsf{then}\ \mathsf{r}\ \mathsf{else}\ \mathsf{s} := \mathsf{cr}[\mathsf{s}]^{\tau\Rightarrow\tau}$. Thus, $\mathbf{tt}r[s]^{\tau\Rightarrow\tau} \hookrightarrow^* r$, while $\mathbf{ff}r[s]^{\tau\Rightarrow\tau} \rightleftarrows^* \mathbf{ff}[s]^{\tau\Rightarrow\tau}r \hookrightarrow^* \{[s]^{\tau\Rightarrow\tau}\}^{\tau\Rightarrow\tau} \hookrightarrow s$.

8 Conclusion, Discussion and Future Work

In this paper we have defined System I, a proof system for propositional logic, where isomorphic propositions have the same proofs.

8.1 Non-terminating extension

As mentioned in the introduction, the choice of rules is subtle. Indeed, as well known, the strong normalisation of simply typed lambda calculus is not a very robust property: minor modifications of typing or reduction rules can lead to non-terminating calculi, see for instance [18]. In System I, we have the rule $(\textsc{dist}_{\mathsf{app}})$ to deal with the equivalence $A \Rightarrow (B \wedge C) \equiv (A \Rightarrow B) \wedge (A \Rightarrow C)$, and we could have also considered a rule such as $\pi_A(rs) \rightleftarrows \pi_{B\Rightarrow A}(r)s$ [13]. However, adding such a rule leads to a non-terminating calculus, as shown by the following example. Let $\delta = \lambda x^{(\tau\Rightarrow\tau)\wedge\tau}.\pi_{\tau\Rightarrow\tau}(x)\pi_\tau(x) : ((\tau \Rightarrow \tau) \wedge \tau) \Rightarrow \tau$, $\delta' = \delta((z^{\tau\Rightarrow\tau\Rightarrow\tau}y^\tau) \times y^\tau) : \tau$, and $\Omega = \delta((z^{\tau\Rightarrow\tau\Rightarrow\tau}y^\tau) \times \delta') : \tau$. Then, we have

$$
\begin{aligned}
&\Omega\\
&\hookrightarrow \pi_{\tau\Rightarrow\tau}((zy) \times \delta')\pi_\tau((zy) \times \delta') \hookrightarrow \pi_{\tau\Rightarrow\tau}((zy) \times \delta')\delta' = \pi_{\tau\Rightarrow\tau}((zy) \times (\delta((zy) \times y)))\delta'\\
&\rightleftarrows \pi_{\tau\Rightarrow\tau}((zy) \times (\delta(zy)y))\delta' \rightleftarrows \pi_{\tau\Rightarrow\tau}((z \times (\delta(zy)))y)\delta' \overset{(\textsc{wrong-rule})}{\rightleftarrows^*} \pi_{\tau\Rightarrow\tau}((z \times (\delta(zy)))\delta')y\\
&\rightleftarrows \pi_{\tau\Rightarrow\tau}((z\delta') \times (\delta(zy)\delta'))y \rightleftarrows \pi_{\tau\Rightarrow\tau}((z\delta') \times (\delta((zy) \times \delta')))y = \pi_{\tau\Rightarrow\tau}((z\delta') \times \Omega)y
\end{aligned}
$$

8.2 Other Related Work

Apart from the related work already discussed in the introduction, in a work by Garrigue and Aït-Kaci [20], the selective λ-calculus has been presented, where only the isomorphism

$$(A \Rightarrow B \Rightarrow C) \equiv (B \Rightarrow A \Rightarrow C). \tag{5}$$

has been treated, which is complete with respect to the function type. In System I we also consider the conjunction, and hence four isomorphisms. Isomorphism (5) is a consequence of currification and commutation, that is $A \wedge B \equiv B \wedge A$ and $(A \wedge B) \Rightarrow C \equiv (A \Rightarrow B \Rightarrow C)$.

The selective λ-calculus includes labellings to identify which argument is being used at each time. Moreover, by considering the Church encoding of pairs, isomorphism (5) implies isomorphism (1) (commutativity of $\wedge$). However, their proposal is different to ours. In particular, we track the term by its type, which is a kind of labelling, but when two terms have the same type, then we leave the system to non-deterministically choose any proof. One of our main novelties is, indeed, the non-deterministic projector. However, we can also get back determinism, by encoding a labelling, as discussed in Section 7, or by dropping some isomorphisms (namely, associativity and commutativity of conjunction).

8.3 Towards more connectives

A subtle question is how to add a neutral element of the conjunction, which will imply more isomorphisms, e.g. $A \wedge \top \equiv A$, $A \Rightarrow \top \equiv \top$ and $\top \Rightarrow A \equiv A$. Adding the equation $\top \Rightarrow \top \equiv \top$ would make it possible to derive $(\lambda x^\top.xx)(\lambda x^\top.xx) : \top$, however this term is

not the classical Ω, it is typed by $\top$, and imposing some restrictions on the beta reduction, it could be forced not to reduce to itself but to discard its argument. For example: "If $A \equiv \top$, then $(\lambda x^A.r)s \hookrightarrow r[\star/x]$", where $\star : \top$ is the introduction rule of $\top$.

8.4 Eta-expansion rule

In [15] we have given an implementation embedded in Haskell of an extended fragment of the system as presented in [13], which is an early version of System I. In such an implementation, we have added some rules in order to have only introductions as normal forms. For example, "If $s : B$ then $(\lambda x^A.\lambda y^B.r)s \hookrightarrow \lambda x^A.((\lambda y^B.r)s)$. Such a rule, among others introduced in this implementation, is a particular case of a more general η-expansion rule. Indeed, with the rule" "If $t : A \Rightarrow B$ then $t \hookrightarrow \lambda x^A.tx$" we can derive $(\lambda x^A.\lambda y^B.r)s \hookrightarrow \lambda z^A.(\lambda x^A.\lambda y^B.r)sz \rightleftarrows^* \lambda z^A.(\lambda x^A.\lambda y^B.r)zs \hookrightarrow \lambda z^A.((\lambda y^B.r[z/x])s)$.

Indeed, we conjecture that System I extended with an η-expansion rule would lead to a system where there is no closed elimination term in normal form. Such an extension is left for future work.

References

1. Pablo Arrighi and Alejandro Díaz-Caro. A System F Accounting for Scalars. *Logical Methods in Computer Science*, 8(1:11), 2012.
2. Pablo Arrighi, Alejandro Díaz-Caro, and Benoît Valiron. The Vectorial Lambda-Calculus. *Information and Computation*, 254(1):105–139, 2017.
3. Pablo Arrighi and Gilles Dowek. Linear-algebraic lambda-calculus: higher-order, encodings, and confluence. In Andrei Voronkov, editor, *Proceedings of RTA 2008*, volume 5117 of *LNCS*, pages 17–31, 2008.
4. Pablo Arrighi and Gilles Dowek. Lineal: A linear-algebraic lambda-calculus. *Logical Methods in Computer Science*, 13(1:8), 2017.
5. Gérard Boudol. Lambda-Calculi for (Strict) Parallel Functions. *Information and Computation*, 108(1):51–127, 1994.
6. Kim B. Bruce, Roberto Di Cosmo, and Giuseppe Longo. Provable isomorphisms of types. *Mathematical Structures in Computer Science*, 2(2):231–247, 1992.
7. Antonio Bucciarelli, Thomas Ehrhard, and Giulio Manzonetto. A Relational Semantics for Parallelism and Non-Determinism in a Functional Setting. *Annals of Pure and Applied Logic*, 163(7):918–934, 2012.
8. Thierry Coquand and Gérard Huet. The Calculus of Constructions. *Information and Computation*, 76(2–3):95–120, 1988.
9. Ugo de'Liguoro and Adolfo Piperno. Non Deterministic Extensions of Untyped λ-calculus. *Information and Computation*, 122(2):149–177, 1995.
10. Mariangiola Dezani-Ciancaglini, Ugo de'Liguoro, and Adolfo Piperno. A filter model for concurrent λ-calculus. *SIAM Journal on Computing*, 27(5):1376–1419, 1998.
11. Roberto Di Cosmo. *Isomorphisms of types: from λ-calculus to information retrieval and language design*. Progress in Theoretical Computer Science. Birkhauser, 1995.
12. Roberto Di Cosmo. A short survey of isomorphisms of types. *Mathematical Structures in Computer Science*, 15(5):825–838, 2005.
13. Alejandro Díaz-Caro and Gilles Dowek. Non determinism through type isomorphism. In Delia Kesner and Petrucio Viana, editors, *Proceedings of LSFA 2012*, volume 113 of *EPTCS*, pages 137–144, 2013.
14. Alejandro Díaz-Caro and Gilles Dowek. Typing quantum superpositions and measurement. In Carlos Martín-Vide, Roman Neruda, and Miguel A. Vega-Rodríguez, editors, *Proceedings of TPNC 2017*, volume 10687 of *LNCS*, pages 281–293, 2017.

15 Alejandro Díaz-Caro and Pablo E. Martínez López. Isomorphisms considered as equalities: Projecting functions and enhancing partial application through an implementation of λ^+. In *Proceedings of the 27th Symposium on the Implementation and Application of Functional Programming Languages*, IFL '15, pages 9:1–9:11. ACM, 2015.

16 Alejandro Díaz-Caro and Barbara Petit. Linearity in the non-deterministic call-by-value setting. In Luke Ong and Ruy de Queiroz, editors, *Proceedings of WoLLIC 2012*, volume 7456 of *LNCS*, pages 216–231, 2012.

17 Gilles Dowek, Thérèse Hardin, and Claude Kirchner. Theorem proving modulo. *Journal of Automated Reasoning*, 31(1):33–72, 2003.

18 Gilles Dowek and Ying Jiang. On the expressive power of schemes. *Information and Computation*, 209:1231–1245, 2011.

19 Gilles Dowek and Benjamin Werner. Proof normalization modulo. *The Journal of Symbolic Logic*, 68(4):1289–1316, 2003.

20 Jacques Garrigue and Hassan Aït-Kaci. The typed polymorphic label-selective λ-calculus. In *Proceedings of POPL 1994*, ACM SIGPLAN, pages 35–47, 1994.

21 Herman Geuvers, Robbert Krebbers, James McKinna, and Freek Wiedijk. Pure Type Systems without Explicit Contexts. In Karl Crary and Marino Miculan, editors, *Proceedings of LFMTP 2010*, volume 34 of *EPTCS*, pages 53–67, 2010.

22 Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and types*. Cambridge University Press, 1989.

23 Per Martin-Löf. *Intuitionistic type theory*. Studies in proof theory. Bibliopolis, 1984.

24 Michele Pagani and Simona Ronchi Della Rocca. Linearity, non-determinism and solvability. *Fundamental Informaticae*, 103(1–4):173–202, 2010.

25 Jonghyun Park, Jeongbong Seo, Sungwoo Park, and Gyesik Lee. Mechanizing Metatheory without Typing Contexts. *Journal of Automated Reasoning*, 52(2):215–239, 2014.

26 Mikael Rittri. Retrieving library identifiers via equational matching of types. In *Proceedings of CADE 1990*, volume 449 of *LNCS*, pages 603–617, 1990.

27 The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. `http://homotopytypetheory.org/book`, Institute for Advanced Study, 2013.

28 Lionel Vaux. The algebraic lambda calculus. *Mathematical Structures in Computer Science*, 19(5):1029–1059, 2009.

A Detailed proofs of Section 4

▶ **Lemma 4.3** (Substitution). *If $r : A$, $s : B$, and $x \in \mathcal{V}_B$, then $r[s/x] : A$.*

Proof. By structural induction on r.

- Let $r = x$. By Lemma 4.2, $A \equiv B$, thus $s : A$. We have $x[s/x] = s$, so $x[s/x] : A$.
- Let $r = y$, with $y \neq x$. We have $y[s/x] = y$, so $y[s/x] : A$.
- Let $r = \lambda y^C.r'$. By Lemma 4.2, $A \equiv C \Rightarrow D$, with $r' : D$. By the induction hypothesis $r'[s/x] : D$, and so, by rule $(\Rightarrow_i)$, $\lambda y^C.r'[s/x] : C \Rightarrow D$. Since $\lambda y^C.r'[s/x] = (\lambda y^C.r')[s/x]$, using rule $(\equiv)$, $(\lambda y^C.r')[s/x] : A$.
- Let $r = r_1 r_2$. By Lemma 4.2, $r_1 : C \Rightarrow A$ and $r_2 : C$. By the induction hypothesis $r_1[s/x] : C \Rightarrow A$ and $r_2[s/x] : C$, and so, by rule $(\Rightarrow_e)$, $(r_1[s/x])(r_2[s/x]) : A$. Since $(r_1[s/x])(r_2[s/x]) = (r_1 r_2)[s/x]$, we have $(r_1 r_2)[s/x] : A$.
- Let $r = r_1 \times r_2$. By Lemma 4.2, $r_1 : A_1$ and $r_2 : A_2$, with $A \equiv A_1 \wedge A_2$. By the induction hypothesis $r_1[s/x] : A_1$ and $r_2[s/x] : A_2$, and so, by rule $(\wedge_i)$, $(r_1[s/x]) \times (r_2[s/x]) : A_1 \wedge A_2$. Since $(r_1[s/x]) \times (r_2[s/x]) = (r_1 \times r_2)[s/x]$, using rule $(\equiv)$, we have $(r_1 \times r_2)[s/x] : A$.
- Let $r = \pi_A(r')$. By Lemma 4.2, $r' : A \wedge C$. Hence, by the induction hypothesis, $r'[s/x] : A \wedge C$. Hence, by rule $\wedge_e$, $\pi_A(r'[s/x]) : A$. Since $\pi_A(r'[s/x]) = \pi_A(r')[s/x]$, we have $\pi_A(r')[s/x] : A$. ◀

▶ **Theorem 4.4** (Subject reduction). *If $r : A$ and $r \hookrightarrow s$ or $r \rightleftarrows s$ then $s : A$.*

Proof. By induction on the rewrite relation.

- (COMM): If $r \times s : A$, then by Lemma 4.2, $A \equiv A_1 \wedge A_2 \equiv A_2 \wedge A_1$, with $r : A_1$ and $s : A_2$. Then, $s \times r : A_2 \wedge A_1 \equiv A$.
- (ASSO):
  ($^\rightarrow$) If $(r \times s) \times t : A$, then by Lemma 4.2, $A \equiv (A_1 \wedge A_2) \wedge A_3 \equiv A_1 \wedge (A_2 \wedge A_3)$, with $r : A_1$, $s : A_2$ and $t : A_3$. Then, $r \times (s \times t) : A_1 \wedge (A_2 \wedge A_3) \equiv A$.
  ($_\leftarrow$) Analogous to ($^\rightarrow$).
- (DIST$_\lambda$):
  ($^\rightarrow$) If $\lambda x^B.(r \times s) : A$, then by Lemma 4.2, $A \equiv (B \Rightarrow (C_1 \wedge C_2)) \equiv ((B \Rightarrow C_1) \wedge (B \Rightarrow C_2))$, with $r : C_1$ and $s : C_2$. Then, $\lambda x^B.r \times \lambda x^B.s : (B \Rightarrow C_1) \wedge (B \Rightarrow C_2) \equiv A$.
  ($_\leftarrow$) If $\lambda x^B.r \times \lambda x^B.s : A$, then by Lemma 4.2, $A \equiv ((B \Rightarrow C_1) \wedge (B \Rightarrow C_2)) \equiv (B \Rightarrow (C_1 \wedge C_2))$, with $r : C_1$ and $s : C_2$. Then, $\lambda x^B.(r \times s) : B \Rightarrow (C_1 \wedge C_2) \equiv A$.
- (DIST$_{\mathsf{app}}$):
  ($^\rightarrow$) If $(r \times s)t : A$, then by Lemma 4.2, $r \times s : B \Rightarrow A$, and $t : B$. Hence, by Lemma 4.2 again, $B \Rightarrow A \equiv C_1 \wedge C_2$, and so by Lemma 2.11, $A \equiv A_1 \wedge A_2$, with $r : B \Rightarrow A_1$ and $s : B \Rightarrow A_2$. Then, $rt \times st : A_1 \wedge A_2 \equiv A$.
  ($_\leftarrow$) If $rt \times st : A$, then by Lemma 4.2, $A \equiv A_1 \wedge A_2$ with $r : B \Rightarrow A_1$, $s : B' \Rightarrow A_2$, $t : B$ and $t : B'$. By Lemma 4.1, $B \equiv B'$. Then $(r \times s)t : A_1 \wedge A_2 \equiv A$.
- (CURRY):
  ($^\rightarrow$) If $rst : A$, then by Lemma 4.2, $r : B \Rightarrow C \Rightarrow A \equiv (B \wedge C) \Rightarrow A$, $s : B$ and $t : C$. Then, $r(s \times t) : A$.
  ($_\leftarrow$) If $r(s \times t) : A$, then by Lemma 4.2, $r : (B \wedge C) \Rightarrow A \equiv (B \Rightarrow C \Rightarrow A)$, $s : B$ and $t : C$. Then $rst : A$.
- (β): If $(\lambda x^B.r)s : A$, then by Lemma 4.2, $\lambda x^B.r : B \Rightarrow A$, and by Lemma 4.2 again, $r : A$. Then by Lemma 4.3, $r[s/x^B] : A$.
- (π): If $\pi_B(r \times s) : A$, then by Lemma 4.2, $A \equiv B$, and so, by rule ($\equiv$), $r : A$.
- Contextual closure: Let $t \to r$, where $\to$ is either $\rightleftarrows$ or $\hookrightarrow$.
  - Let $\lambda x^B.t \to \lambda x^B.r$: If $\lambda x^B.t : A$, then by Lemma 4.2, $A \equiv (B \Rightarrow C)$ and $t : C$, hence by the induction hypothesis, $r : C$ and so $\lambda x^B.r : B \Rightarrow C \equiv A$.
  - Let $ts \to rs$: If $ts : A$ then by Lemma 4.2, $t : B \Rightarrow A$ and $s : B$, hence by the induction hypothesis, $r : B \Rightarrow A$ and so $rs : A$.
  - Let $st \to st$: If $st : A$ then by Lemma 4.2, $s : B \Rightarrow A$ and $t : B$, hence by the induction hypothesis $r : B$ and so $sr : A$.
  - Let $t \times s \to r \times s$: If $t \times s : A$ then by Lemma 4.2, $A \equiv A_1 \wedge A_2$, $t : A_1$, and $s : A_2$, hence by the induction hypothesis, $r : A_1$ and so $r \times s : A_1 \wedge A_2 \equiv A$.
  - Let $s \times t \to s \times r$: Analogous to previous case.
  - Let $\pi_B(t) \to \pi_B(r)$: If $\pi_B(t) : A$ then by Lemma 4.2, $A \equiv B$ and $t : B \wedge C$, hence by the induction hypothesis $r : B \wedge C$. Therefore, $\pi_B(r) : B \equiv A$. ◀

B Detailed proofs of Section 5

B.1 Detailed proofs of Section 5.2

▶ **Lemma 5.11.** *If $r \times s \rightleftarrows^* t$ then either*

1. *$t = u \times v$ where either*
   a. *$u \rightleftarrows^* t_{11} \times t_{21}$ and $v \rightleftarrows^* t_{12} \times t_{22}$ with $r \rightleftarrows^* t_{11} \times t_{12}$ and $s \rightleftarrows^* t_{21} \times t_{22}$, or*
   b. *$v \rightleftarrows^* w \times s$ with $r \rightleftarrows^* u \times w$, or any of the three symmetric cases, or*
   c. *$r \rightleftarrows^* u$ and $s \rightleftarrows^* v$, or the symmetric case.*

2. $t = \lambda x^A.a$ *and* $a \rightleftarrows^* a_1 \times a_2$ *with* $r \rightleftarrows^* \lambda x^A.a_1$ *and* $s \rightleftarrows^* \lambda x^A.a_2$.
3. $t = av$ *and* $a \rightleftarrows^* a_1 \times a_2$*, with* $r \rightleftarrows^* a_1 v$ *and* $s \rightleftarrows^* a_2 v$.

Proof. By a double induction, first on $M(t)$ and then on the length of the relation $\rightleftarrows^*$. Consider an equivalence proof $r \times s \rightleftarrows^* t' \rightleftarrows t$ with a shorter proof $r \times s \rightleftarrows^* t'$. By the second induction hypothesis, the term t' has the form prescribed by the lemma. We consider the three cases and in each case, the possible rules transforming t' in t.

1. Let $r \times s \rightleftarrows^* u \times v \rightleftarrows t$. The possible equivalences from $u \times v$ are
   - $t = u' \times v$ or $u \times v'$ with $u \rightleftarrows u'$ and $v \rightleftarrows v'$, and so the term t is in case 1.
   - Rules (COMM) and (ASSO) preserve the conditions of case 1.
   - $t = \lambda x^A.(u' \times v')$, with $u = \lambda x^A.u'$ and $v = \lambda x^A.v'$. By the first induction hypothesis (since $M(u) < M(t)$ and $M(v) < M(t)$), either
     a. $u \rightleftarrows^* w_{11} \times w_{21}$ and $v \rightleftarrows^* w_{12} \times w_{22}$, by the first induction hypothesis, $w_{ij} \rightleftarrows^* \lambda x^A.t_{ij}$ for $i = 1, 2$ and $j = 1, 2$, with $u' \rightleftarrows^* t_{11} \times t_{21}$ and $v' \rightleftarrows^* t_{12} \times t_{22}$, so $u' \times v' \rightleftarrows^* t_{11} \times t_{12} \times t_{21} \times t_{22}$. Hence, $r \rightleftarrows^* \lambda x^A.(t_{11} \times t_{12})$ and $s \rightleftarrows^* \lambda x^A.(t_{21} \times t_{22})$, and hence the term t is in case 2.
     b. $v \rightleftarrows^* w \times s$ and $r \rightleftarrows^* u \times w$. Since $v \rightleftarrows^* \lambda x^A.v'$, by the first induction hypothesis, $w \rightleftarrows^* \lambda x^A.t_1$ and $s \rightleftarrows^* \lambda x^A.t_2$, with $v' \rightleftarrows^* t_1 \times t_2$. Hence, $r \rightleftarrows^* \lambda x.(u' \times t_1)$, and hence the term t is in case 2.
     c. $r \rightleftarrows^* \lambda x^A.u'$ and $s \rightleftarrows^* \lambda x^A.v$, and hence the term t is in case 2.

     (the symmetric cases are analogous).
   - $t = (u' \times v')t'$, with $u = u't'$ and $v = v't'$. By the first induction hypothesis (since $M(u) < M(t)$ and $M(v) < M(t)$), either
     a. $u \rightleftarrows^* w_{11} \times w_{21}$, $v \rightleftarrows^* w_{12} \times w_{22}$, $r \rightleftarrows^* w_{11} \times w_{12}$, and $s \rightleftarrows^* w_{21} \times w_{22}$. By the first induction hypothesis (since $M(w_{ij}) < M(t)$), $w_{ij} \rightleftarrows^* t_{ij}t'$, for $i = 1, 2$ and $j = 1, 2$, where $u' \rightleftarrows^* t_{11} \times t_{21}$, $v' \rightleftarrows^* t_{12} \times t_{22}$, $r \rightleftarrows^* w_{11} \times w_{12}$ and $s \rightleftarrows^* w_{21} \times w_{22}$. Therefore, $u \rightleftarrows^* t_{11}t' \times t_{21}t'$ and $v \rightleftarrows^* t_{12}t' \times t_{22}t'$ with $r \rightleftarrows^* (t_{11} \times t_{12})t'$ and $s \rightleftarrows^* (t_{21} \times t_{22})t'$, and hence the term t is in case 3.
     b. $v't' \rightleftarrows^* w \times s$ and $r \rightleftarrows^* u't' \times w$. By the first induction hypothesis on $v't' \rightleftarrows^* w \times s$ (since $M(w) < M(t)$ and $M(s) < M(t)$), we have $w \rightleftarrows^* t_1 t'$ and $s \rightleftarrows^* t_2 t'$ with $t_1 \times t_2 \rightleftarrows^* v'$. Therefore, $v \rightleftarrows^* t_1 t' \times t_2 t'$ with $r \rightleftarrows^* (u \times t_1)t'$ and $s \rightleftarrows^* t_2 t'$, and $u' \times v' \rightleftarrows^* u' \times t_1 \times t_2$, hence the term t is in case 3.
     c. $r \rightleftarrows^* u't'$ and $s \rightleftarrows^* v't'$, and hence we are in case 3.

     (the symmetric cases are analogous).
2. Let $r \times s \rightleftarrows^* \lambda x^A.a \rightleftarrows t$, with $a \rightleftarrows^* a_1 \times a_2$, $r \rightleftarrows^* \lambda x^A.a_1$, and $s \rightleftarrows^* \lambda x^A.a_2$. Hence, possible equivalences from $\lambda x.a$ to t are
   - $t = \lambda x^A.a'$ with $a \rightleftarrows^* a'$, hence $a' \rightleftarrows^* a_1 \times a_2$, and so the term t is in case 2.
   - $t = \lambda x^A.u \times \lambda x^A.v$, with $a_1 \times a_2 \rightleftarrows^* a = u \times v$. Hence, by the first induction hypothesis (since $M(a) < M(t)$), either
     a. $a_1 \rightleftarrows^* u$ and $a_2 \rightleftarrows^* v$, and so $r \rightleftarrows^* \lambda x^A.u$ and $s \rightleftarrows^* \lambda x^A.v$, or
     b. $v \rightleftarrows^* t_1 \times t_2$ with $a_1 \rightleftarrows^* u \times t_1$ and $a_2 \rightleftarrows^* t_2$, and so $\lambda x^A.v \rightleftarrows^* \lambda x.t_1 \times \lambda x^A.t_2$, $r \rightleftarrows^* \lambda x^A.u \times \lambda x^A.t_1$ and $s \rightleftarrows^* \lambda x^A.t_2$, or
     c. $u \rightleftarrows^* t_{11} \times t_{21}$ and $v \rightleftarrows^* t_{12} \times t_{22}$ with $a_1 \rightleftarrows^* t_{11} \times t_{12}$ and $a_2 \rightleftarrows^* t_{21} \times t_{22}$, and so $\lambda x^A.u \rightleftarrows^* \lambda x^A.t_{11} \times \lambda x^A.t_{21}$, $\lambda x.v \rightleftarrows^* \lambda x^A.t_{12} \times \lambda x^A.t_{22}$, $r \rightleftarrows^* \lambda x^A.t_{11} \times \lambda x^A.t_{12}$ and $s \rightleftarrows^* \lambda x^A.t_{21} \times \lambda x^A.t_{22}$.

     (the symmetric cases are analogous), and so the term t is in case 1.
3. Let $r \times s \rightleftarrows^* aw \rightleftarrows t$, with $a \rightleftarrows^* a_1 \times a_2$, $r \rightleftarrows^* a_1 w$, and $s \rightleftarrows^* a_2 w$. The possible equivalences from aw to t are

- $t = a'w$ with $a \rightleftarrows^* a'$, hence $a' \rightleftarrows^* a_1 \times a_2$, and so the term t is in case 3.
- $t = aw'$ with $w \rightleftarrows^* w'$ and so the term t is in case 3.
- $t = uw \times vw$, with $a_1 \times a_2 \rightleftarrows^* a = u \times v$. Hence, by the first induction hypothesis (since $M(a) < M(t)$), either
  a. $a_1 \rightleftarrows^* u$ and $a_2 \rightleftarrows^* v$, and so $r \rightleftarrows^* uw$ and $s \rightleftarrows^* vw$, or
  b. $v \rightleftarrows^* t_1 \times t_2$ with $a_1 \rightleftarrows^* u \times t_1$ and $a_2 \rightleftarrows^* t_2$, and so $vw \rightleftarrows^* t_1 w \times t_2 w$, $r \rightleftarrows^* uw \times t_1 w$ and $s \rightleftarrows^* t_2 w$, or
  c. $u \rightleftarrows^* t_{11} \times t_{21}$ and $v \rightleftarrows^* t_{12} \times t_{22}$ with $a_1 \rightleftarrows^* t_{11} \times t_{12}$ and $a_2 \rightleftarrows^* t_{21} \times t_{22}$, and so $uw \rightleftarrows^* t_{11} w \times t_{21} w$, $vw \rightleftarrows^* t_{12} w \times t_{22} w$, $r \rightleftarrows^* t_{11} w \times t_{12} w$ and $s \rightleftarrows^* t_{21} w \times t_{22} w$.

  (the symmetric cases are analogous), and so the term t is in case 1.
- $t = a'(v \times w)$ with $a = a'v$, thus $a'v = a \rightleftarrows^* a_1 \times a_2$. Hence, by the first induction hypothesis, $a' \rightleftarrows^* a'_1 \times a'_2$, with $a_1 \rightleftarrows^* a'_1 v$ and $a_2 \rightleftarrows^* a'_2 v$. Therefore, $r \rightleftarrows^* a'_1(v \times w)$ and $s \rightleftarrows^* a'_2(v \times w)$, and so the term t is in case 3. ◀

▶ **Lemma 5.12.** *If $r_1 \times r_2 \rightleftarrows^* s \hookrightarrow t$, there exists u_1, u_2 such that $t \rightleftarrows^* u_1 \times u_2$ and either $(r_1 \rightsquigarrow u_1$ and $r_2 \rightsquigarrow u_2)$, or $(r_1 \rightsquigarrow u_1$ and $r_2 \rightleftarrows^* u_2)$, or $(r_1 \rightleftarrows^* u_1$ and $r_2 \rightsquigarrow u_2)$.*

Proof. By induction on $M(r_1 \times r_2)$. By Lemma 5.11, s is either a product, an abstraction or an application with the conditions given in the lemma. The different terms s reducible by $\hookrightarrow$ are
- $(\lambda x^A.a)s'$ that reduces by the (β) rule to $a[s'/x]$.
- $s_1 \times s_2$, $\lambda x^A.a$, as', with a reduction in the subterm s_1, s_2, a, or s'.

Notice that rule (π) cannot apply since $s \not\rightleftarrows^* \pi_C(s')$.

We consider each case:
- $s = (\lambda x^A.a)s'$ and $t = a[s'/x]$. Using twice Lemma 5.11, we have $a \rightleftarrows^* a_1 \times a_2$, $r_1 \rightleftarrows^* (\lambda x^A.a_1)s'$ and $r_2 \rightleftarrows^* (\lambda x^A.a_2)s'$. Since $t \rightleftarrows^* a_1[s'/x] \times a_2[s'/x]$, we take $u_1 = a_1[s'/x]$ and $u_2 = a_2[s'/x]$.
- $s = s_1 \times s_2$, $t = t_1 \times s_2$ or $t = s_1 \times t_2$, with $s_1 \hookrightarrow t_1$ and $s_2 \hookrightarrow t_2$. We only consider the first case since the other is analogous. One of the following cases happen
  (a) $r_1 \rightleftarrows^* w_{11} \times w_{21}$, $r_2 \rightleftarrows^* w_{12} \times w_{22}$, $s_1 = w_{11} \times w_{12}$ and $s_2 = w_{21} \times w_{22}$. Hence, by the induction hypothesis, either $t_1 = w'_{11} \times w_{12}$, or $t_1 = w_{11} \times w'_{12}$, or $t_1 = w'_{11} \times w'_{12}$, with $w_{11} \hookrightarrow w'_{11}$ and $w_{12} \hookrightarrow w'_{12}$. We take, in the first case $u_1 = w'_{11} \times w_{21}$ and $u_2 = w_{12} \times w_{22}$, in the second case $u_1 = w_{11} \times w_{21}$ and $u_2 = w'_{12} \times w_{22}$, and in the third $u_1 = w'_{11} \times w_{21}$ and $u_2 = w'_{12} \times w_{22}$.
  (b) We consider two cases, since the other two are symmetric.
    - $r_1 \rightleftarrows^* s_1 \times w$ and $s_2 \rightleftarrows^* w \times r_2$, in which case we take $u_1 = t_1 \times w$ and $u_2 = r_2$.
    - $r_2 \rightleftarrows^* w \times s_2$ and $s_1 = r_1 \times w$. Hence, by the induction hypothesis, either $t_1 = r'_1 \times w$, or $t_1 = r_1 \times w'$ or $t_1 = r'_1 \times w'$, with $r_1 \hookrightarrow r'_1$ and $w \hookrightarrow w'$. We take, in the first case $u_1 = r'_1$ and $u_2 = w \times s_2$, in the second case $u_1 = r_1$ and $u_2 = w' \times s_2$, and in the third case $u_1 = r'_1$ and $u_2 = w' \times s_2$.

  (c) $r_1 \rightleftarrows^* s_1$ and $r_2 \rightleftarrows^* s_2$, in which case we take $u_1 = t_1$ and $u_2 = s_2$.
- $s = \lambda x^A.a$, $t = \lambda x^A.t'$, and $a \hookrightarrow t'$, with $a \rightleftarrows^* a_1 \times a_2$ and $s \rightleftarrows^* \lambda x^A.a_1 \times \lambda x^A.a_2$. Therefore, by the induction hypothesis, there exists u'_1, u'_2 such that either $(a_1 \rightsquigarrow u'_1$ and $a_2 \rightsquigarrow u'_2)$, or $(a_1 \rightleftarrows^* u'_1$ and $a_2 \rightsquigarrow u'_2)$, or $(a_1 \rightsquigarrow u'_1$ and $a_2 \rightleftarrows^* u'_2)$. Therefore, we take $u_1 = \lambda x^A.u'_1$ and $u_2 = \lambda x^A.u'_2$.
- $s = as'$, $t = t's'$, and $a \hookrightarrow t'$, with $a \rightleftarrows^* a_1 \times a_2$ and $s \rightleftarrows^* a_1 s' \times a_2 s'$. Therefore, by the induction hypothesis, there exists u'_1, u'_2 such that either $(a_1 \rightsquigarrow u'_1$ and $a_2 \rightsquigarrow u'_2)$, or $(a_1 \rightleftarrows^* u'_1$ and $a_2 \rightsquigarrow u'_2)$, or $(a_1 \rightsquigarrow u'_1$ and $a_2 \rightleftarrows^* u'_2)$. Therefore, we take $u_1 = u'_1 s'$ and $u_2 = u'_2 s'$.

- $s = as'$, $t = at'$, and $s' \hookrightarrow t'$, with $a \rightleftarrows^* a_1 \times a_2$ and $s \rightleftarrows^* a_1s' \times a_2s'$. By Lemma 5.11 several times, one the following cases happen
  (a) $a_1s' \rightleftarrows^* w_{11}s' \times w_{12}s'$, $a_2s' \rightleftarrows^* w_{21}s' \times w_{22}s'$, $r_1 \rightleftarrows^* w_{11}s' \times w_{21}s'$ and $r_2 \rightleftarrows^* w_{12}s' \times w_{22}s'$. We take $u_1 \rightleftarrows^* (w_{11} \times w_{21})t'$ and $r_2 \rightleftarrows^* (w_{12} \times w_{22})t'$.
  (b) $a_2s' \rightleftarrows^* w_1s' \times w_2s'$, $r_1 \rightleftarrows^* a_1s' \times w_2s'$ and $r_2 \rightleftarrows^* w_2s'$. So we take $u_1 = (a_1 \times w_1)t'$ and $u_2 = w_2t'$, the symmetric cases are analogous.
  (c) $r_1 \rightleftarrows^* a_1s'$ and $r_2 \rightleftarrows^* a_2s'$, in which case we take $u_1 = a_1t'$ and $u_2 = a_2t'$ the symmetric case is analogous. ◀

B.2 Detailed proofs of Section 5.5

▶ **Lemma 5.27** (Adequacy of product). *If $r \in [\![A]\!]$ and $s \in [\![B]\!]$, then $r \times s \in [\![A \wedge B]\!]$.*

Proof. We need to prove that $K^{\tau}_{A\wedge B}[r \times s] \in \mathsf{SN}$. We proceed by induction on the number of projections in $K^{\tau}_{A\wedge B}$. Since the hole of $K^{\tau}_{A\wedge B}$ has type $A \wedge B$, and $K^{\tau}_{A\wedge B}[t]$ has type τ for any $t : A$ there is at least one projection.

As $r \in [\![A]\!]$, for any elimination context K'^{τ}_A, we have $K'^{\tau}_A[r] \in \mathsf{SN}$, but then if $A \equiv B_1 \Rightarrow \cdots \Rightarrow B_n \Rightarrow C$, we also have $K''^{\tau}_C[rt_1 \dots t_n] \in \mathsf{SN}$, thus $rt_1 \dots t_n \in [\![C]\!]$. Similarly, since $s \in [\![B]\!]$, $st_1 \dots t_n$ is reducible.

We prove that $K'^{\tau}_C[\pi_C(rt_1 \dots t_n \times st_1 \dots t_n)] \in \mathsf{SN}$ by showing, more generally, that if r' and s' are two reducts of $rt_1 \dots t_n$ and $st_1 \dots t_n$, then $K'^{\tau}_C[\pi_C(r' \times s')] \in \mathsf{SN}$. For this, we show that all its one step reducts are in SN, by induction on $|K'^{\tau}_C| + |r'| + |s'|$.

- If the reduction takes place in one of the terms in $\mathcal{T}(K'^{\tau}_C)$, in r', or in s', we apply the induction hypothesis.
- Otherwise, the reduction is a (π) reduction of $\pi_C(r' \times s')$, that is, $r' \times s' \rightleftarrows^* v \times w$, the reduct is v, and we need to prove $K'^{\tau}_C[v] \in \mathsf{SN}$. By Lemma 5.11, we have either:
  - $v \rightleftarrows^* r_1 \times s_1$, with $r' \rightleftarrows^* r_1 \times r_2$ and $s' \rightleftarrows^* s_1 \times s_2$. In such a case, by Lemma 5.25, v is the product of two reducible terms, so since there is one projection less than in $K^{\tau}_{A\wedge B}$, the first induction hypothesis applies.
  - $v \rightleftarrows^* r' \times s_1$, with $s' \rightleftarrows^* s_1 \times s_2$. In such a case, by Lemma 5.25, v is the product of two reducible terms, so since there is one projection less than in $K^{\tau}_{A\wedge B}$, the first induction hypothesis applies.
  - $v \rightleftarrows^* r_1 \times s'$, with $r' \rightleftarrows^* r_1 \times r_2$. In such a case, by Lemma 5.25, v is the product of two reducible terms, so since there is one projection less than in $K^{\tau}_{A\wedge B}$, the first induction hypothesis applies.
  - $v \rightleftarrows^* r'$, in which case, $C \equiv A$, and since $r \in [\![A]\!]$, we have $K'^{\tau}_A[r'] \rightleftarrows^* K'^{\tau}_A[v] \in \mathsf{SN}$.
  - $v \rightleftarrows^* r_1$ with $r' \rightleftarrows^* r_1 \times r_2$, in which case, since $r \in [\![A]\!]$, we have $K'^{\tau}_C[\pi_C(r')] \in \mathsf{SN}$ and $K'^{\tau}_C[\pi_C(r')] \rightsquigarrow K'^{\tau}_C[v]$ hence $K'^{\tau}_C[v] \in \mathsf{SN}$.
  - $v \rightleftarrows^* s'$, in which case, $C \equiv B$, and since $s \in [\![B]\!]$, we have $K'^{\tau}_B[s'] \rightleftarrows^* K'^{\tau}_B[v] \in \mathsf{SN}$.
  - $v \rightleftarrows^* s_1$ with $s' \rightleftarrows^* s_1 \times s_2$, in which case, since $s \in [\![B]\!]$, we have $K'^{\tau}_C[\pi_C(s')] \in \mathsf{SN}$ and $K'^{\tau}_C[\pi_C(s')] \rightsquigarrow K'^{\tau}_C[v]$ hence $K'^{\tau}_C[v] \in \mathsf{SN}$. ◀

▶ **Lemma 5.28** (Adequacy of abstraction). *If for all $t \in [\![A]\!]$, $r[t/x] \in [\![B]\!]$, then $\lambda x^A.r \in [\![A \Rightarrow B]\!]$.*

Proof. We proceed by induction on $M(r)$.

If $r \rightleftarrows^* r_1 \times r_2$, by Lemma 4.2, $B \equiv B_1 \wedge B_2$ with $r_1 : B_1$ and $r_2 : B_2$. and so by Lemma 4.3, $r_1[t/x] : B_1$ and $r_2[t/x] : B_2$. Since $r[t/x] \in [\![B]\!]$, we have $r_1[t/x] \times r_2[t/x] \in [\![B]\!]$. By Lemma 5.25, $r_1[t/x] \in [\![B_1]\!]$ and $r_2[t/x] \in [\![B_2]\!]$. By the induction hypothesis, $\lambda x^A.r_1 \in$

$[\![A \Rightarrow B_1]\!]$ and $\lambda x^A.r_2 \in [\![A \Rightarrow B_2]\!]$, then by Lemma 5.27, $\lambda x^A.r \rightleftarrows^* \lambda x^A.r_1 \times \lambda x^A.r_2 \in [\![(A \Rightarrow B_1) \wedge (A \Rightarrow B_2)]\!]$, and by Lemma 5.23, $[\![(A \Rightarrow B_1) \wedge (A \Rightarrow B_2)]\!] = [\![A \Rightarrow B]\!]$.

If $r \not\rightleftarrows^* r_1 \times r_2$, we need to prove that for any elimination context $K^\tau_{A\Rightarrow B}$, we have $K^\tau_{A\Rightarrow B}[\lambda x^A.r] \in \mathsf{SN}$.

Since r and all the terms in $\mathcal{T}(K^\tau_{A\Rightarrow B})$ are reducible, then they are in SN, by Lemma 5.24. We proceed by induction on $|K^\tau_{A\Rightarrow B}|+|r|$ to show that all the one step reducts of $K^\tau_{A\Rightarrow B}[\lambda x^A.r]$ are in SN. Since r is not a product, the only one step reducts are the following.

- If the reduction takes place in one of the terms in $\mathcal{T}(K^\tau_{A\Rightarrow B})$ or r, we apply the induction hypothesis.
- If $K^\tau_{A\Rightarrow B}[\lambda x^A.r] = K'^\tau_B[(\lambda x^A.r)s]$ and it reduces to $K'^\tau_B[r[s/x]]$, as $r[s/x] \in [\![B]\!]$, we have $K'^\tau_B[r[s/x]] \in \mathsf{SN}$. ◀

▶ **Theorem 5.30** (Adequacy)**.** *If $r : A$, then for all σ adequate, we have $\sigma r \in [\![A]\!]$.*

Proof. By induction on r.

- If r is a variable $x \in \mathcal{V}_A$, then, since σ is adequate, we have $\sigma r \in [\![A]\!]$.
- If r is a product $s \times t$, then by Lemma 4.2, $s : B$, $t : C$, and $A \equiv B \wedge C$, then by the induction hypothesis, $\sigma s \in [\![B]\!]$ and $\sigma t \in [\![C]\!]$. By Lemma 5.27, $(\sigma s \times \sigma t) \in [\![B \wedge C]\!]$, hence by Lemma 5.23, $\sigma r \in [\![A]\!]$.
- If r is a projection $\pi_A(s)$, then by Lemma 4.2, $s : A \wedge B$, and by the induction hypothesis, $\sigma s \in [\![A \wedge B]\!]$. By Lemma 5.25, $\pi_A(\sigma s) \in [\![A]\!]$, hence $\sigma r \in [\![A]\!]$.
- If r is an abstraction $\lambda x^B.s$, with $s : C$, then by Lemma 4.2, $A \equiv B \Rightarrow C$, hence by the induction hypothesis, for all σ, and for all $t \in [\![B]\!]$, $(\sigma s)[t/x] \in [\![C]\!]$. Hence, by Lemma 5.28, $\lambda x^B.\sigma s \in [\![B \Rightarrow C]\!]$, hence, by Lemma 5.23, $\sigma r \in [\![A]\!]$.
- If r is an application st, then by Lemma 4.2, $s : B \Rightarrow A$ and $t : B$, then by the induction hypothesis, $\sigma s \in [\![B \Rightarrow A]\!]$ and $\sigma t \in [\![B]\!]$. Hence, by Lemma 5.26, we have $\sigma r = \sigma s \sigma t \in [\![A]\!]$. ◀

λ!-calculus, Intersection Types, and Involutions

Alberto Ciaffaglione
Department of Mathematics, Computer Science and Physics, University of Udine, Italy
alberto.ciaffaglione@uniud.it

Pietro Di Gianantonio
Department of Mathematics, Computer Science and Physics, University of Udine, Italy
pietro.digianantonio@uniud.it

Furio Honsell
Department of Mathematics, Computer Science and Physics, University of Udine, Italy
furio.honsell@uniud.it

Marina Lenisa
Department of Mathematics, Computer Science and Physics, University of Udine, Italy
marina.lenisa@uniud.it

Ivan Scagnetto
Department of Mathematics, Computer Science and Physics, University of Udine, Italy
ivan.scagnetto@uniud.it

Abstract

Abramsky's *affine combinatory algebras* are models of *affine combinatory logic*, which refines standard combinatory logic in the direction of Linear Logic. Abramsky introduced various universal models of computation based on affine combinatory algebras, consisting of *partial involutions* over a suitable formal language of moves, in order to discuss *reversible computation* in a *Geometry of Interaction* setting. We investigate partial involutions from the point of view of the model theory of λ!-calculus. The latter is a refinement of the standard λ-calculus, corresponding to affine combinatory logic. We introduce *intersection type systems* for the λ!-calculus, by extending standard intersection types with a $!_u$-operator. These induce affine combinatory algebras, and, via suitable quotients, models of the λ!-calculus. In particular, we introduce an intersection type system for assigning *principal types* to λ!-terms, and we state a correspondence between the partial involution interpreting a combinator and the principal type of the corresponding λ!-term. This analogy allows for explaining as *unification* between principal types the somewhat awkward *linear application* of involutions arising from Geometry of Interaction.

2012 ACM Subject Classification Theory of computation → Program semantics; Theory of computation → Linear logic

Keywords and phrases Affine Combinatory Algebra, Affine Lambda-calculus, Intersection Types, Geometry of Interaction

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.15

Funding Work supported by the Italian departmental research project "LambdaBridge" (D.R.N. 427/2018 of 03/08/2018, University of Udine).

1 Introduction

In [1], S. Abramsky discusses *reversible computation* in a game-theoretic setting. In particular, he introduces various kinds of reversible *pattern-matching automata* whose behaviour can be described in a *finitary* way as partial injective functions, actually *involutions*, over a suitable language of moves. These automata are *universal* in that they yield *affine combinatory algebras*.

© Alberto Ciaffaglione, Pietro Di Gianantonio, Furio Honsell, Marina Lenisa, and Ivan Scagnetto;
licensed under Creative Commons License CC-BY

The crucial notion is that of application between automata, or between partial involutions. This is essentially the application between *history-free strategies* used in *Game Semantics*, which itself stems from Girard's *Execution Formula*, or Abramsky's *symmetric feedback* [3]. The former was introduced by J. Y. Girard [15, 16] in the context of "Geometry of Interaction" (GoI) to model, in a language-independent way, the fine semantics of *Linear Logic*. Constructions similar to the Combinatory Algebra of partial involutions, introduced in [1], appear in various papers by S. Abramsky, *e.g.* [2, 4], and are special cases of a general categorical paradigm explored by E. Haghverdi [17] (Sections 5.3, 6), called "Abramsky's Programme". This Programme amounts to defining a *linear* λ*-algebra* starting from a *GoI Situation* in a "traced symmetric monoidal category".

In the present paper we carry out an analysis of Abramsky's algebras from the point of view of the model theory of λ-calculus. It is a follow up to [8, 9], where only the purely linear and affine fragments of Affine Combinatory Algebras are considered. Here we extend the *involutions-as-principal types/GoI application-as-resolution* analogy introduced in [8, 9] to the full calculus, offering a new perspective on Girard's Geometry of Interaction and how its reversible dynamics arises.

More specifically, we focus on the notion of *affine combinatory logic*, its λ-calculus counterpart, the $\lambda!$-calculus, and their models, *i.e. affine-combinatory algebras* and *affine-combinatory* λ*-algebras*[1].

Our approach stems from realizing the existence of a *structural analogy*, introduced in [9], which to our knowledge had not been hitherto pointed out in the literature, between the *Geometry of Interaction* interpretation of a λ-term in Abramsky's model of partial involutions and the *principal type* of that term, with respect to an *intersection type discipline* for the $\lambda!$*-calculus*. This we termed *involutions-as-types* analogy. Intersection types originated in [6] and have been utilised in discussing games in a different approach also in [13, 14]. In particular, we define an algorithm which, given a principal type of a λ-term, reads off the partial involution corresponding to the interpretation of that term. Thus we show that the principal type of an affine λ-term provides a characterisation of the partial involution interpreting the term in Abramsky's model. Conversely, we show how to extract a "principal type" from *any* partial involution, possibly not corresponding to any λ-term.

The *involutions-as-types* analogy is very fruitful. It allows for simply explaining as a *unification* between principal types the somewhat awkward *linear application* between involutions used in [1], deriving from the notion of application used throughout the literature on GoI and Game Semantics. We call this the "GoI application-as-resolution of principal types" analogy, or more simply the *application-as-resolution* analogy. The overall effect of linear application amounts, indeed, to *unifying* the left-hand side of the principal type of the operator with the principal type of the operand, and applying the resulting substitution to the right hand side of the operator. Hence, the notion of application between partial involutions, corresponding to λ-terms M and N, can be explained as computing the involution corresponding to the principal type of MN, given the principal types of M and N. Actually this unification mechanism works even if the types are not the types of any concrete λ-term.

Our analysis, therefore, unveils three conceptually independent, but ultimately equivalent, accounts of *application* in the λ-calculus: β*-reduction*, the GoI application of involutions based on symmetric feedback/Girard's *Execution Formula*, and *resolution* of principal types. In order to check our theoretical results, we have implemented in Erlang [12, 5] application of involutions, as well as compilation of λ-terms as combinators and their interpretation as involutions.

[1] This notion was originally introduced by D. Scott for the standard λ-calculus as the appropriate notion of categorical model for the calculus, see Section 5.2 of [7].

Synopsis. In Section 2, we collect the definitions of affine combinatory logic, λ!-calculus, affine combinatory algebra, and λ-algebra. In Section 3, we recall Abramsky's combinatory algebra of partial involutions. In Section 4, we provide an intersection type system for the λ!-calculus, and we study its properties. In Section 5, we define a correspondent principal type assignment system for assigning only the most general types to λ!-terms. In Section 6, we explore the relationships between principal types and partial involutions, giving evidence to the involutions-as-types analogy. Concluding remarks appear in Section 7. The Web Appendix [19] includes the detailed Erlang programs implementing compilations and effective operations on partial involutions.

2 Affine Combinatory Logic and the λ!-calculus

In this section, we collect the notions of affine combinatory logic, λ!-calculus, affine combinatory algebra, and λ-algebra. These notions amount to the Linear Logic refinements of the corresponding standard notions.

▶ **Definition 1** (Affine Combinatory Logic). *The language of* affine combinatory logic $\mathbf{CL}^!$ *includes variables $x, y, \ldots$, distinguished constants (combinators) $B, C, I, K, W, D, \delta, F$, and it is closed under application and promotion, i.e.:*

$$\frac{M \in \mathbf{CL}^! \quad N \in \mathbf{CL}^!}{MN \in \mathbf{CL}^!} \qquad \frac{M \in \mathbf{CL}^!}{!M \in \mathbf{CL}^!}$$

Combinators satisfy the following equations for all terms of $\mathbf{CL}^!$, M, N, P (we associate $\cdot$ to the left and we assume ! to have order of precedence greater than $\cdot$):

$$\begin{array}{llll} BMNP = M(NP) & IM = M & CMNP = (MP)N & KMN = M \\ WM!N = M!N!N & \delta !M = !!M & D!M = M & F!M!N = !(MN) \end{array}$$

The λ!-calculus is the λ-calculus counterpart of affine combinatory logic:

▶ **Definition 2** (Affine λ!-calculus). *The language $\mathbf{\Lambda}^!$ of the* affine λ!-calculus *is inductively defined from variables $x, y, z, \ldots$, and it is closed under the following formation rules:*

$$\frac{M \in \mathbf{\Lambda}^! \quad N \in \mathbf{\Lambda}^!}{MN \in \mathbf{\Lambda}^!} \qquad \frac{M \in \mathbf{\Lambda}^!}{!M \in \mathbf{\Lambda}^!} \qquad \frac{M \in \mathbf{\Lambda}^! \quad \mathcal{O}_!(x, M)}{\lambda x.M \in \mathbf{\Lambda}^!} \qquad \frac{M \in \mathbf{\Lambda}^! \quad \mathcal{M}_!(x, M)}{\lambda !x.M \in \mathbf{\Lambda}^!},$$

where $\mathcal{O}_!(x, M)$ means that the variable x appears free in M at most once, *and it is not in the scope of a !, while $\mathcal{M}_!(x, M)$ means that the variable x appears free in M* at least once.

The reduction rules of the λ!-calculus are the restrictions of the standard β-rule and ξ-rule to linear abstractions, the pattern-β-reduction *rule, which defines the behaviour of the !-pattern abstraction operator, the str!-structural rule, and the ξ!-rule, namely:*

$$(\beta)\ \ (\lambda x.M)N \to M[N/x] \qquad (\beta!)\ \ (\lambda !x.M)!N \to M[N/x]$$

$$(\xi)\ \ \frac{M \to N \quad \lambda x.M, \lambda x.N \in \mathbf{\Lambda}^!}{\lambda x.M \to \lambda x.N} \qquad (str!)\ \ \frac{M \to N}{!M \to !N} \qquad (\xi!)\ \ \frac{M \to N}{\lambda !x.M \to \lambda !x.N}$$

All the remaining rules are as in the standard case. We denote by $=_{\lambda^!}$ the induced congruence relation.

The λ!-calculus introduced above is quite similar to the calculus introduced in [18], the only differences being that our calculus is affine, while the one in [18] is linear, moreover reduction under the scope of a !-operator is forbidden in [18], while we allow for it.

▶ **Proposition 3.** *Well-formedness in $\mathbf{\Lambda}^!$ is preserved under λ!-conversion. The corresponding reduction λ!-calculus is Church-Rosser.*

The correspondence between affine combinatory logic and λ!-calculus is formalized below.

▶ **Definition 4.** *We define two homomorphisms w.r.t. ! and application:*
(i) $(\ )_{\lambda^!} : \mathbf{CL}^! \to \mathbf{\Lambda}^!$*, given a term M of $\mathbf{CL}^!$, yields the term of $\mathbf{\Lambda}^!$ obtained from M by replacing, in place of each combinator, the corresponding $\mathbf{\Lambda}^!$-term as follows*

$(B)_{\lambda^!} = \lambda xyz.x(yz)$	$(W)_{\lambda^!} = \lambda x!y.x!y!y$
$(C)_{\lambda^!} = \lambda xyz.(xz)y$	$(D)_{\lambda^!} = \lambda !x.x$
$(I)_{\lambda^!} = \lambda x.x$	$(\delta)_{\lambda^!} = \lambda !x.!!x$
$(K)_{\lambda^!} = \lambda xy.x$	$(F)_{\lambda^!} = \lambda !x!y.!(xy)$

(ii) $(\ )_{CL^!} : \mathbf{\Lambda}^! \to \mathbf{CL}^!$*, given a term M of the λ!-calculus, replaces each λ-abstraction by a λ^*-abstraction. Terms with λ^*-abstractions amount to $\mathbf{CL}^!$-terms via the* Abstraction Operation *defined below.*

▶ **Definition 5** (Abstraction Operation)**.** *The following algorithm is defined by induction on $ML \in \mathbf{CL}^!$:*

$$\lambda^* x.x = I \quad \lambda^* !x.x = D \quad \lambda^* !x.!x = F(!I)$$

$$\lambda^* x.MN = \begin{cases} C(\lambda^* x.M)N & \text{if } x \in FV(M), \\ BM(\lambda^* x.N) & \text{if } x \in FV(N). \end{cases}$$

$$\lambda^* !x.MN = W(C(BB(\lambda^* !x.M))(\lambda^* !x.N)) \qquad \lambda^* !x.!M = B(F(!\lambda^* !x.M))\delta, \text{ for } M \not\equiv x.$$

Notice that, alternatively, $\lambda^* !x.MN$ can be defined permuting C and B, *i.e.* $\lambda^* !x.MN = W(B(C\lambda^* !x.M)(\lambda^* !x.N))$. This ambivalence is a source of problems which ultimately makes affine combinatory algebras fail to be λ-algebras (see Definition 9 below).

▶ **Theorem 6** (Affine Abstraction Theorem)**.** *For all terms $\lambda x.M, N \in \mathbf{CL}^!$,*
$(\lambda^* x.M)N = M[N/x]$ *and* $(\lambda^* !x.M)!N = M[N/x]$.

The semantical counterpart of $\mathbf{CL}^!$ is the notion of affine combinatory algebra:

▶ **Definition 7** (Affine Combinatory Algebra, [1])**.** *An* affine combinatory algebra *(*ACA*)*, $\mathcal{A} = (A, \cdot, !)$ *is an applicative structure* $(A, \cdot)$ *with a unary (injective) operation !, and combinators $B, C, I, K, W, D, \delta, F$ satisfying the following equations: for all $x, y, z \in A$,*

$Bxyz = x(yz)$	$Ix = x$	$Cxyz = (xz)y$	$Kxy = x$
$Wx!y = x!y!y$	$\delta !x = !!x$	$D!x = x$	$F!x!y = !(xy)$.

▶ **Definition 8.** *Given an affine combinatory algebra $\mathcal{A} = (A, \cdot, !)$, we define the set of affine combinatory terms $\mathcal{T}(\mathcal{A})$ as the extension of $\mathbf{CL}^!$ with constants c_a for any point $a \in A$.*

▶ **Definition 9** (Affine λ-algebra)**.** *An ACA $\mathcal{A}$ is an* affine λ-algebra *if, for all $M, N \in \mathcal{T}(\mathcal{A})$,*

$$\vdash (M)_{\lambda^!} =_{\lambda^!} (N)_{\lambda^!} \implies [\![M]\!]_{\mathcal{A}} = [\![N]\!]_{\mathcal{A}} ,$$

where $[\![\ ]\!]_{\mathcal{A}}$ denotes the natural interpretation of terms in $\mathcal{T}(\mathcal{A})$ over the ACA $\mathcal{A}$.

One can prove that there exists an equivalent characterisation of the notion of affine λ-algebra via *equations* involving combinators. For lack of space, we omit this characterisation.

3 The Model of Partial Involutions

In [1], Abramsky exploits the connection between automata and strategies, and he introduces various reversible universal models of computation. Building on earlier work, *e.g.* [4, 17], Abramsky defines models arising from *Geometry of Interaction Situations*, consisting of history-free strategies. He discusses a model of *partial injections* and $\mathcal{P}$, its substructure consisting of *partial involutions*. Partial involutions are defined over a suitable language of moves, and they can be endowed with a structure of an affine combinatory algebra:

▶ **Definition 10** (The Model of Partial Involutions $\mathcal{P}$).

(i) T_Σ, *the language of* moves, *is defined by the signature* $\Sigma_0 = \{\epsilon\}$, $\Sigma_1 = \{l, r\}$, $\Sigma_2 = \{< \, , \, >\}$ *(where* Σ_i *is the set of constructors of arity* i*); terms* $r(x)$ *are* output words, *while terms* $l(x)$ *are* input words *(often denoted simply by* rx *and* lx*);*

(ii) $\mathcal{P}$ *is the set of* partial involutions *over* T_Σ, *i.e. the set of all partial injective functions* $f : T_\Sigma \rightharpoonup T_\Sigma$ *such that* $f(u) = v \Leftrightarrow f(v) = u$*;*

(iii) *the operation of* replication *is defined by* $!f = \{(< t, u >, < t, v >) \mid t \in T_\Sigma \wedge (u, v) \in f\}$*;*

(iv) *the notion of* linear application *is defined by* $f \cdot g = f_{rr} \cup (f_{rl}; g; (f_{ll}; g)^*; f_{lr})$, *where* $f_{ij} = \{(u, v) | (i(u), j(v)) \in f\}$, *for* $i, j \in \{r, l\}$ *(see Fig. 1), where "$;$" denotes postfix composition.*

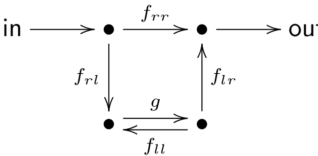


Figure 1 Flow of control in executing $f \cdot g$.

Following [1], we make a slight abuse of notation and assume that T_Σ contains pattern variables for terms. The intended meaning will be clear from the context. In the sequel, we will use the notation $u_1 \leftrightarrow v_1, \ldots, u_n \leftrightarrow v_n$, for $u_1, \ldots, u_n, v_1, \ldots, v_n \in T_\Sigma$, to denote the graph of the (finite) partial involution f defined by $\forall i.(f(u_i) = v_i \ \wedge \ f(v_i) = u_i)$. Again, following [1], we will use the above notation in place of a more automata-like presentation of the partial involution.

▶ **Proposition 11** ([1], Th.5.1). *$\mathcal{P}$ can be endowed with the structure of an affine combinatory algebra, $(\mathcal{P}, \cdot, !)$, where combinators are defined by the following partial involutions:*

B	:	$r^3x \leftrightarrow lrx \ , \ l^2x \leftrightarrow rlrx \ , \ rl^2x \leftrightarrow r^2lx$	I	:	$lx \leftrightarrow rx$
C	:	$l^2x \leftrightarrow r^2lx \ , \ lrlx \leftrightarrow rlx \ , \ lr^2x \leftrightarrow r^3x$	K	:	$lx \leftrightarrow r^2x$
F	:	$l\langle x, ry\rangle \leftrightarrow r^2\langle x, y\rangle \ , \ l\langle x, ly\rangle \leftrightarrow rl\langle x, y\rangle$	δ	:	$l\langle\langle x, y\rangle, z\rangle \leftrightarrow r\langle x, \langle y, z\rangle\rangle$
W	:	$r^2x \leftrightarrow lr^2x \ , \ l^2\langle x, y\rangle \leftrightarrow rl\langle lx, y\rangle \ , \ lrl\langle x, y\rangle \leftrightarrow rl\langle rx, y\rangle$	D	:	$l\langle \epsilon, x\rangle \leftrightarrow rx$.

4 The !Intersection Type Discipline for the λ!-calculus

In this section, we introduce an intersection type system for the λ!-calculus, and we study its properties. In particular, we prove that subject reduction holds up-to an appropriate relation on types, while subject conversion holds when we consider some restrictions of β-reduction, *e.g. lazy reduction* or *closed reduction*. Reduction is *lazy* when it is *not* applied under a λ-abstraction, it is *closed* when only β-redexes with *closed* arguments can be reduced.

Types in this system include a $\multimap$-constructor, a $!_u$-constructor, for u ranging over indexes, and a $\wedge$-constructor. We introduce the $!_u$-constructor in order to establish a connection between types and partial involutions, this correspondence will be formally stated in Section 6. From a more intuitive point of view, the $!_u$-constructors are associated to the parts of terms that can be potentially replicated: free or bound variables that can be used several times inside a term; terms that are used as function arguments and that can be replicated inside the function. The indexes u are mainly used to distinguish one (potential) replica from the other. Moreover one can observe that there are dependencies among replications, for example, the replication of a term leads also to the replication of the several instances of the same variable that the term may contain; indexes are also used to keep track of these dependencies among replications.

For technical reasons, we include also a !-constructor without index. It essentially behaves as $!_\epsilon$, however they are dealt with differently in the !-introduction rule. This choice allows us to maintain the correspondence between types and the interpretations of combinators in the algebra of partial involutions, as we will see in Sections 5 and 6 below.

▶ **Definition 12** (!Intersection Types). *We introduce the following set of types:*

$$(\mathit{Type} \ni)\ \tau, \sigma ::= \alpha \mid \tau \multimap \sigma \mid !\tau \mid !_u\tau \mid \tau \wedge \sigma$$

where α denotes a type variable in $TVar$, and $u, v \in T_\Sigma[IVar]$, where $IVar$ is a set of index variables ranged over by $i, j, \ldots$.

▶ **Definition 13** (!Intersection Type System). *The* !intersection type system *for the λ!-calculus derives judgements $\Gamma; \Delta \Vdash M : \tau$, where $\tau \in \mathit{Type}$ and*

- *the* linear *environment Γ is a set $x_1 : \sigma_1, \ldots, x_m : \sigma_m$;*
- *the* non-linear *environment Δ is a set $!x'_1 : \tau_1, \ldots, !x'_n : \tau_n$, with $\tau_1, \ldots, \tau_n$ having a bang $(!, !_u)$ as main connective;*
- *$dom(\Gamma) \cap dom(\Delta) = \emptyset$;*
- *each variable in Γ occurs at most once, while multiple occurrences of the same variable are possible in Δ.*

The rules for assigning !intersection types *are the following:*

$$\frac{}{x : \tau; \langle\rangle \vdash x : \tau}\ (ax_1) \qquad \frac{}{\langle\rangle; !x :!\tau \vdash x : \tau}\ (ax_2)$$

$$\frac{\langle\rangle; \Delta_1 \vdash M : \tau_1 \quad \ldots \quad \langle\rangle; \Delta_n \vdash M : \tau_n \qquad u_1, ..., u_n\ distinct}{\langle\rangle; \widehat{!_{u_1}}\Delta_1, \ldots, \widehat{!_{u_n}}\Delta_n \vdash !M :!_{u_1}\tau_1 \wedge \ldots \wedge !_{u_n}\tau_n}\ (!)$$

$$\frac{\Gamma; \Delta \vdash M : \sigma \multimap \tau \quad \Gamma'; \Delta' \vdash N : \sigma \quad dom(\Gamma) \cap dom(\Gamma') = \emptyset \quad dom(\Gamma, \Gamma') \cap dom(\Delta, \Delta') = \emptyset}{\Gamma, \Gamma'; \Delta \wedge \Delta' \vdash MN : \tau}\ (app)$$

$$\frac{\Gamma, x : \sigma; \Delta \vdash M : \tau \quad \mathcal{O}_!(x, M)}{\Gamma; \Delta \vdash \lambda x.M : \sigma \multimap \tau}\ (\lambda_L) \qquad \frac{\Gamma; \Delta \vdash M : \tau \quad x, \sigma\ fresh}{\Gamma; \Delta \vdash \lambda x.M : \sigma \multimap \tau}\ (\lambda_A)$$

$$\frac{\Gamma; \Delta, !x : \sigma_1, \ldots, !x : \sigma_n \vdash M : \tau \quad x \notin dom(\Delta)}{\Gamma; \Delta \vdash \lambda !x.M : (\sigma_1 \wedge \cdots \wedge \sigma_n) \multimap \tau}\ (\lambda_!)$$

where

- $\widehat{!}_u(!x_1 : \tau_1, \ldots, !x_n : \tau_n) \equiv !x_1 : \widehat{!}_u\tau_1, \ldots, !x_n : \widehat{!}_u\tau_n$,
  where $\widehat{!}_u\tau$ *is defined by:* $\begin{cases} \widehat{!}_u(!\tau) = !_u\tau \\ \widehat{!}_u(!_v\tau) = !_{\langle u,v\rangle}\tau \end{cases}$
- $\Delta \wedge \Delta'$ *is the environment* Δ'' *defined by:* $!x : \tau \in \Delta''$ *if and only if*
  $!x : \tau \in \Delta$ *and* $x \notin dom(\Delta')$,
  or $!x : \tau \in \Delta'$ *and* $x \notin dom(\Delta)$,
  or $\tau \equiv !_{lu}\tau'$ *and* $!x : !_u\tau \in \Delta$ *and* $x \in dom(\Delta')$,
  or $\tau \equiv !_{ru}\tau'$ *and* $!x : !_u\tau' \in \Delta'$ *and* $x \in dom(\Delta)$.
- *In rule (app), by abuse of notation, the subtype* σ *in the type* $\sigma \multimap \tau$ *assigned to* M *and the type* σ *assigned to* N *coincide only up-to equating occurrences of* $!$ *and* $!_\epsilon$.

A few remarks on the definition above are in order.

- The !-constructor on types with no index can be eliminated by replacing the axiom (ax_2) with an alternative axiom (ax_2') having form: $\dfrac{}{\langle\rangle; !x :!_\epsilon\tau \vdash x : \tau}\ (\mathrm{ax}_2')$
  With this last axiom one obtains a type system quite similar to the original one (and in some sense simpler): the derivable types will differ just by the presence of some extra ϵ symbols; all the properties for the type system stated in the paper still hold. However, the interpretation of the combinatory constants induced by the alternative type system will be different from the one given on the algebra of partial involutions $\mathcal{P}$. To preserve the interpretation of combinatory constants, we preferred to use a more ad-hoc type system.
- In derivations of type judgements, the order in which hypotheses are derived is relevant, *i.e.* the *nature* of the $\wedge$-operator is *non-commutative* and *non-associative*. In the present type assignment system, we take into account the order of hypotheses by prefixing $\{l, r\}$-tags in !-indexes, when we merge non-linear environments in the (app)-rule. As a consequence, tags describe the structure of a $\wedge$-type, and $\wedge$ is considered to be commutative and associative in the present system. Of course, we could have equivalently omitted $\{l, r\}$-tags in merging non-linear environments and explicitly used a non-commutative and non-associative $\wedge$-operator, both in the environments and in the assigned types. Our choice is justified by the fact that this presentation of the type assignment system makes the correspondence between types and partial involutions more direct (see Section 6).

Some immediate consequences on the shape of judgments derivable in the type system are the following:

▶ **Lemma 14.** *If* $\Gamma; \Delta \vdash M : \tau$*, then*

(i) *for all* $!x : \sigma \in \Delta$, σ *is in the form* $!\tau$ *or* $!_u\tau$

(ii) $FV(M) = dom(\Gamma) \cup dom(\Delta)$ *and* $x \in dom(\Gamma) \Leftrightarrow \mathcal{O}_!(x, M)$.

Intuitively, the λ!-terms which are typable in the system are essentially the terms which strongly normalize to terms not containing *forever stuck applications*, in the sense of Lemma 15(iv) below.

▶ **Lemma 15.**

(i) *If* $\Gamma; \Delta \vdash \lambda !x.M : \tau$*, then there exist* $\sigma, \sigma' \in Type$ *such that* $\tau = \sigma \multimap \sigma'$*, and* σ *is a !-type or a* $\wedge$*-type.*

(ii) *If* $\Gamma; \Delta \vdash N : \tau$*, where* τ *is a !-type or a* $\wedge$*-type, then* N *is not a* λ*-abstraction.*

(iii) *If* $\Gamma; \Delta \vdash !N : \tau$*, then* $\Gamma = \emptyset$ *and* τ *is a !-type or a* $\wedge$*-type.*

(iv) *Terms which contain subterms of the shape* $(\lambda !x.M)(\lambda y.N)$*, or* $(\lambda !x.M)(\lambda !y.N)$*,* $!MN$ *are not typable.*

Proof. The proof of items (i), (ii), (iii), is straightforward, by induction on derivations. Item (iv) follows from the previous ones. ◀

Now we study subject reduction and conversion properties of the system. We will show that subject reduction holds up-to an equivalence relation on types, while subject conversion fails in general. However, there are two interesting cases in which subject conversion holds, either up-to an equivalence relation on types or in its full form, respectively: when β-reduction is *lazy*, *i.e.* it is *not* applied under a λ-abstraction, or when we allow for reducing only β-redexes whose argument is a *closed* λ-term.

Intuitively, the reasons why an equivalence relation is necessary for ensuring subject reduction in the general case and subject conversion in the lazy case are the following.

- Intuitively, in β-reducing, the order in which hypotheses are used to type the resulting term is different from the order in which these are used to type the starting term. Therefore subject reduction holds only up a suitably renaming of tags in $\wedge$-types and environments. This is related to the behaviour of the !-operator in history-free game models, where also appropriate equivalences renaming !-indexes are required.
- The behaviour of the !-index ϵ is peculiar in the present type system: namely, it may happen that a redex is typable with a type τ where $\langle\epsilon, u\rangle$ or $\langle u, \epsilon\rangle$ appear as !-indexes, but the reduced term is typable only with a type τ' differing from τ because the indexes $\langle\epsilon, u\rangle$ or $\langle u, \epsilon\rangle$ are replaced by u. For example, $z :!_{\langle\epsilon,i\rangle}\alpha \vdash (\lambda!x.x)!!z :!_i\alpha$, but $!z :!_i\alpha \vdash !z :!_i\alpha$ and $!z :!_{\langle\epsilon,i\rangle}\alpha \not\vdash !z :!_i\alpha$.

Moreover, notice that subject conversion fails already on the purely affine fragment when we allow for β-reducing under λ-abstractions, *e.g.* we cannot derive $\vdash \lambda xyz.(\lambda w.x)(yz) : \alpha_1 \multimap \alpha_2 \multimap \alpha_3 \multimap \alpha_1$, but only $\vdash \lambda xyz.(\lambda w.x)(yz) : \alpha_1 \multimap (\alpha_2 \multimap \alpha_3) \multimap \alpha_2 \multimap \alpha_1$, but we have $\vdash \lambda xyz.x : \alpha_1 \multimap \alpha_2 \multimap \alpha_3 \multimap \alpha_1$ (see [9] for more details).

To formalize the above facts, we start by introducing the following relation on types:

▶ **Definition 16.**

- *Let $\approx_I$ be the least congruence relation on $T_\Sigma[IVar]$ such that, for any $u \in T_\Sigma[IVar]$, $u \approx_I \langle\epsilon, u\rangle \approx_I \langle u, \epsilon\rangle$, $u \approx_I lu \approx_I ru$, and $\langle u_1, \langle u_2, u_3\rangle\rangle \approx_I \langle\langle u_1, u_2\rangle, u_3\rangle$.*
- *Let $\approx$ be the least congruence relation on types such that, for any permutation p on $T_\Sigma[IVar]$ satifying the condition $\forall u \in T_\Sigma[IVar]\,.\, u \approx_I p(u)$, we have:*

$$(!_{u_1}\sigma_1 \wedge \ldots \wedge !_{u_n}\sigma_n) \multimap \tau \;\approx\; (!_{p(u_1)}\sigma_1 \wedge \ldots \wedge !_{p(u_n)}\sigma_n) \multimap \tau$$

- *For Δ, Δ' non-linear environments, we define $\Delta \approx \Delta'$ if, for each variable x with bindings $!x :!_{u_1}\sigma_1, \ldots, !x :!_{u_n}\sigma_n$ in Δ, there exist a permutation p on $T_\Sigma[IVar]$ and a list of types $\sigma'_1, \ldots, \sigma'_n$ such that*
  - *the binding for x in Δ' are $!x :!_{p(u_1)}\sigma'_1, \ldots, !x :!_{p(u_n)}\sigma'_n$,*
  - *$\forall u \in T_\Sigma[IVar]\,.\, u \approx p(u)$,*
  - *$\forall i.\sigma_i \approx \sigma'_i$.*

▶ **Lemma 17** (Substitution).

(i) *If $x \in FV(M)$ and $\mathcal{O}_!(x, M)$, then*
$\Gamma; \Delta \vdash (\lambda x.M)N : \tau \iff \Gamma; \Delta' \vdash M[N/x] : \tau$, *with* $\Delta' \approx \Delta$.
If N is closed, then $\Delta' = \Delta$.

(ii) $\Gamma; \Delta \vdash (\lambda!x.M)!N : \tau \iff \Gamma; \Delta' \vdash M[N/x] : \tau$, *with* $\Delta' \approx \Delta$.
If N is closed, then $\Delta' = \Delta$.

(iii) *If N is a typable term and $x \notin FV(M)$, then*
$\Gamma;\Delta \vdash (\lambda x.M)N : \tau \iff \Gamma';\Delta' \vdash M : \tau$, *with* $\Gamma' = \Gamma_{|FV(M)}$ *and* $\Delta' \subseteq \Delta$.
If N is closed, then $\Gamma' = \Gamma$ *and* $\Delta' = \Delta$.

Proof.

(i) The thesis follows from (a) and (b) below:

(a) $\Gamma, x:\sigma;\Delta \vdash M : \tau \;\&\; \Gamma';\Delta' \vdash N : \sigma \;\&\; dom(\Gamma) \cap dom(\Gamma') = \emptyset \implies \exists \Delta''.(\Gamma \cup \Gamma';\Delta'' \vdash M[N/x] : \tau \;\&\; (\Delta'' \approx \Delta \wedge \Delta')_{|dom(\Delta)\cap dom(\Delta')})$,
where $(\Delta'' \approx \Delta \wedge \Delta')_{|dom(\Delta)\cap dom(\Delta')}$ means that, for any variable in $dom(\Delta) \cap dom(\Delta')$, the corresponding types in Δ'' and $\Delta \wedge \Delta'$ are $\approx$-equivalent, while for other variables, the corresponding types are equal.

(b) (b) $\Gamma'';\Delta'' \vdash M[N/x] : \tau \;\&\; \Gamma';\Delta' \vdash N : \sigma \implies \exists \Delta.(\Gamma, x:\sigma;\Delta \vdash M : \tau \;\&\; \Gamma'' = \Gamma \cup \Gamma' \;\&\; (\Delta'' \approx \Delta \wedge \Delta')_{|dom(\Delta)\cap dom(\Delta')})$.

Facts (a) and (b) above can be proved by induction on the structure of M.

(ii) The thesis follows from (a) and (b) below:

(a) $\Gamma;\Delta, !x :!_{u_1}\sigma_1, \dots, !x :!_{u_n}\sigma_n \vdash M : \tau \;\&\; \langle\rangle;\Delta_1 \vdash N : \sigma_1 \;\&\; \dots \;\&\; \langle\rangle;\Delta_n \vdash N : \sigma_n$
$\implies \exists \Delta''.(\Gamma;\Delta'' \vdash M[N/x] : \tau \;\&\; \Delta'' \approx \Delta, \widehat{!}_{u_1}\Delta_1, \dots, \widehat{!}_{u_n}\Delta_n)$.

(b) $\Gamma;\Delta'' \vdash M[N/x] : \tau \implies \exists \Delta, \Delta_1, \dots, \Delta_n \,.\, \Delta'' \approx \Delta, \widehat{!}_{u_1}\Delta_1, \dots, \widehat{!}_{u_n}\Delta_n$
$\& \;\Gamma;\Delta, !x :!_{u_1}\sigma_1, \dots, !x :!_{u_n}\sigma_n \vdash M : \tau \;\&\; \langle\rangle;\Delta_1 \vdash N : \sigma_1 \;\&\; \dots \;\&\; \langle\rangle;\Delta_n \vdash N : \sigma_n$.

Facts (a) and (b) above can be proved by induction on the structure of M.

(iii) The thesis follows from a direct analysis of the derivations. ◀

Using the above lemma, one can prove that subject reduction holds up-to-$\preceq$, where $\preceq$ is the relation on types combining $\approx$ with type inclusion:

▶ **Definition 18.**

- *Let $\preceq$ be the least preorder relation on types, compatible with the type constructors and such that:*
  - $(!_{u_1}\sigma_1 \wedge \dots \wedge !_{u_m}\sigma_m) \multimap \tau \;\preceq\; (!_{u_1}\sigma_1 \wedge \dots \wedge !_{u_n}\sigma_n) \multimap \tau$ *when* $m \leq n$
  - $\sigma \approx \tau$ *implies* $\sigma \preceq \tau$
- *For Δ, Δ' non-linear environments, we define $\Delta \preceq \Delta'$ if there exists a non-linear environment Δ'' such that $\Delta'' \subseteq \Delta'$ and for all variables x there exists a one to one correspondence between the types associated to x in Δ and Δ'', and the corresponding types are $\preceq$-related.*

▶ **Theorem 19** (Subject Reduction). *If* $\Gamma;\Delta \vdash M : \tau \;\&\; M \to_\beta M'$, *then* $\exists \Gamma', \Delta', \tau'. \;(\Gamma';\Delta' \vdash M' : \tau' \;\&\; \Gamma' = \Gamma_{|FV(M')} \;\&\; \Delta' \preceq \Delta \;\&\; \tau' \preceq \tau)$.

Proof. The thesis can be proved for one reduction step, $C[(\lambda x.M)N] \to_\beta C[M[N/x]]$, by induction on the context $C[\;]$, using the fact that: for $\sigma' \preceq \sigma$
$\Gamma;\Delta \vdash M : \sigma \iff \Gamma';\Delta' \vdash M : \sigma'$, for $\Delta' \preceq \Delta$. ◀

However, as noticed above, subject conversion fails already on the purely affine fragment, when β-reduction is allowed under λ-abstraction. Nevertheless, if we restrict ourselves to lazy reduction, subject conversion holds up-to-$\sim$, where $\sim$ is the least equivalence including $\preceq$. In what follows, we will denote lazy conversion by $=^L_\beta$.

▶ **Theorem 20** (Lazy Subject Conversion). *If* $\Gamma;\Delta \vdash M : \tau$, $M =^L_\beta M'$ *and M' is typable, then* $\exists \Gamma', \Delta'. \;(\Gamma';\Delta' \vdash M' : \tau \;\&\; \Gamma'_{|FV(M)\cap FV(M')} = \Gamma_{|FV(M)\cap FV(M')} \;\&\; \Delta' \sim \Delta)$, *where $\sim$ denotes the least equivalence relation including $\preceq$.*

Moreover, subject conversion holds exactly (not up-to-$\sim$), in the case in which β-reduction is applied only if the argument is a closed term. In what follows, we will denote closed conversion by $=^C_\beta$.

▶ **Theorem 21** (Closed Subject Conversion). *If* $\Gamma;\Delta \vdash M : \tau$, $M =^C_\beta M'$ *and* M' *is typable, then* $\Gamma;\Delta \vdash M' : \tau$.

Moreover, we have:

▶ **Proposition 22.** *The !Intersection Type System induces an affine combinatory algebra* $(\mathcal{G}, \cdot_\mathcal{G}, !_\mathcal{G})$*, where:*

- $\mathcal{G}$ *is the set of sets of types in Type;*
- *combinatory constants are represented by the sets of types assigned to the* $\Lambda^!$*-terms corresponding to combinators;*
- *for* $\Sigma, \Sigma' \in \mathcal{G}$*, the application is defined by* $\Sigma \cdot_\mathcal{G} \Sigma' = \{\tau \mid \sigma \to \tau \in \Sigma \ \&\ \sigma \in \Sigma'\}$*;*
- $!_\mathcal{G}\Sigma = \{!_u\sigma \mid \sigma \in \Sigma \ \&\ u \in T_\Sigma[IVar]\}$.

5 The !Intersection Principal Type Discipline for the λ!-calculus

In this section, we introduce a type system, where only the most general type schemes are assigned to λ!-terms. As we will show, all type judgements derivable in the intersection type system of Definition 13 can be recovered as instances of judgements derivable in this system, and vice versa all instances of judgements derivable in the principal type system are derivable in the previous one. Moreover, one can prove that for any typable term M there exists a judgement/type of minimal complexity which can be derived/assigned to M, which we call *principal judgement/type*. The crucial rule of the principal type system below is the application rule, where a unification mechanism between the types of the function and the argument is involved. The remaining rules reflect the rules of the type system of Definition 13. As we will see in Section 6, principal types assigned to λ!-terms correspond to partial involutions interpreting the terms in the combinatory algebra of partial involutions.

▶ **Definition 23** (!Intersection Principal Type System). *The* !intersection principal type system *for the λ!-calculus derives judgements* $\Gamma;\Delta \Vdash M : \tau$*, where* $\tau \in$ *Type and*

- *the* linear *environment* Γ *is a set* $x_1 : \sigma_1, \dots, x_m : \sigma_m$*;*
- *the* non-linear *environment* Δ *is a set* $!x'_1 : \tau_1, \dots, !x'_n : \tau_n$*, with* $\tau_1, \dots, \tau_n$ *having a bang* $(!, !_u)$ *as main connective;*
- $dom(\Gamma) \cap dom(\Delta) = \emptyset$*;*
- *each variable in* Γ *occurs at most once, while multiple occurrences of the same variable are possible in* Δ*.*

The rules for assigning principal !intersection types *are the following:*

$$\frac{}{x : \alpha; \langle\rangle \Vdash x : \alpha}\ (ax_1) \qquad \frac{}{\langle\rangle; !x :!\alpha \Vdash x : \alpha}\ (ax_2)$$

$$\frac{\langle\rangle; \Delta_1 \Vdash M : \tau_n \quad \dots \quad \langle\rangle; \Delta_n \Vdash M : \tau_n \quad i_1, \dots i_n \ \textit{fresh}}{\langle\rangle; \widehat{!_{i_1}}\Delta_1, \dots, \widehat{!_{i_n}}\Delta_n \Vdash !M :!_{i_1}\tau_1 \wedge \dots \wedge !_{i_n}\tau_n}\ (!)$$

$$\frac{\begin{array}{c}\Gamma;\Delta \Vdash M : \sigma \quad \Gamma';\Delta' \Vdash N : \tau \quad dom(\Gamma) \cap dom(\Gamma') = \emptyset \quad Var(\Gamma;\Delta,\sigma) \cap Var(\Gamma';\Delta',\tau) = \emptyset \\ U = MGU(\sigma, \alpha \multimap \beta) \quad \alpha,\beta \ \textit{fresh} \quad U' = MGU(U(\alpha),\tau)\end{array}}{(U' \circ U)(\Gamma \cup \Gamma'; \Delta \wedge \Delta') \Vdash MN : (U' \circ U)(\beta)}\ (app)$$

$$\frac{\Gamma, x:\sigma; \Delta \Vdash M:\tau \quad \mathcal{O}_!(x,M)}{\Gamma; \Delta \Vdash \lambda x.M : \sigma \multimap \tau}\ (\lambda_L) \qquad \frac{\Gamma; \Delta \Vdash M:\tau \quad x, \alpha \textit{ fresh}}{\Gamma; \Delta \Vdash \lambda x.M : \alpha \multimap \tau}\ (\lambda_A)$$

$$\frac{\Gamma; \Delta, !x:\sigma_1, \ldots, !x:\sigma_n \vdash M:\tau \quad x \notin dom(\Delta)}{\Gamma; \Delta \vdash \lambda !x.M : (\sigma_1 \wedge \cdots \wedge \sigma_n) \multimap \tau}\ (\lambda_!)$$

where $\widehat{!}_i$ *and* $\Delta \wedge \Delta'$ *are defined as in Definition 13,* $Var(\Gamma; \Delta, \sigma)$ *denotes the set of type and index variables in* Γ, Δ, σ, *and* $(U' \circ U)(\Gamma \cup \Gamma'; \Delta \wedge \Delta')$ *stands for the component wise application of the substitution* $(U' \circ U)$ *to types in the contexts* $\Gamma \cup \Gamma'; \Delta \wedge \Delta'$.

The MGU algorithm is defined as follows:

▶ **Definition 24** ($MGU(\sigma,\tau)$)**.** *Given two types* σ *and* τ, *the partial algorithm* MGU *yields a substitution* U *on types and index variables such that* $U(\sigma) = U(\tau)$.

$$\frac{\alpha \in TVar \quad \alpha \notin \tau}{MGU(\tau,\alpha) = id[\tau/\alpha]} \qquad \frac{\alpha \in TVar \quad \alpha \notin \tau}{MGU(\alpha,\tau) = id[\tau/\alpha]}$$

$$\frac{MGU(u,v) = U'}{MGU(!_u\sigma, !_v\tau) = U \quad MGU(U'(\sigma), U'(\tau)) = U}$$

$$\frac{MGU(\sigma_1,\tau_1) = U_1 \quad MGU(U_1(\sigma_2), U_1(\tau_2)) = U_2}{MGU(\sigma_1 \multimap \sigma_2, \tau_1 \multimap \tau_2) = U_2 \circ U_1}$$

$$\frac{MGU(\sigma_1,\tau_1) = U_1 \quad MGU(U_1(\sigma_2), U_1(\tau_2)) = U_2}{MGU(\sigma_1 \wedge \sigma_2, \tau_1 \wedge \tau_2) = U_2 \circ U_1}$$

$$\frac{i \in IVar \quad i \notin u}{MGU(u,i) = id[u/i]} \qquad \frac{i \in IVar \quad i \notin u}{MGU(i,u) = id[u/i]}$$

$$\frac{MGU(u,v) = U}{MGU(lu, lv) = U} \qquad \frac{MGU(u,v) = U}{MGU(ru, rv) = U} \qquad \frac{MGU(u_1,v_1) = U_1 \quad MGU(u_2,v_2) = U_2}{MGU(\langle u_1,u_2\rangle, \langle v_1,v_2\rangle) = U_2 \circ U_1}$$

where we assume that $!_\epsilon$ *unifies with* $!$.

As it is well known, the above algorithm yields a substitution which factors any other unifier.

One can easily prove the analogue of Lemmata 14 and 15 for the principal type system.

Moreover, an important property of the present system is that types and type judgements have a special shape: each type variable occurs at most twice. As we will see, this is a key observation in relating principal types to partial involutions.

▶ **Definition 25** (Binary Type/Judgement)**.**

- *A* binary type *is a type* $\tau \in Type$ *in which each variable occurs at most twice.*
- *A* binary judgement *is a judgement* $\Gamma; \Delta \Vdash M : \tau$ *in which each variable occurs at most twice.*

▶ **Lemma 26.** *If* $\Gamma; \Delta \Vdash M : \tau$, *then* $\Gamma; \Delta \Vdash M : \tau$ *is a binary judgement.*

Proof. By induction on derivations. ◀

In general, a λ-term M can be assigned different types in a given environment. However, there exists a *minimal* judgment w.r.t. the complexity of types, assigning a type to M, which we call *principal judgement*. For example $\Vdash \lambda !x.!!x : !_{<i,j>}\alpha \multimap !_i !_j \alpha$ is the principal judgement (type) for $\lambda !x.!!x$, but we can also derive $\Vdash \lambda !x.!!x : !_{<i,j_1>}\alpha_1 \wedge !_{<i,j_2>}\alpha_2 \multimap !_i(!_{j_1}\alpha_1 \wedge !_{j_2}\alpha_2)$; namely, using the $\wedge$-rule, we can replicate a !-type more times. In the following definition, we introduce a relation on types formalizing this.

▶ **Definition 27.** *Let $\leq$ be the least reflexive and transitive relation on types defined by:*

$$\frac{}{\sigma \leq \sigma \wedge \tau} \qquad \frac{}{\sigma \leq \tau \wedge \sigma} \qquad \frac{\sigma \leq \sigma' \quad \tau \leq \tau'}{\sigma \multimap \tau \leq \sigma' \multimap \tau'} \qquad \frac{\sigma \leq \sigma'}{!_u\sigma \leq !_u\sigma'} \qquad \frac{\sigma \leq \sigma' \quad \tau \leq \tau'}{\sigma \wedge \tau \leq \sigma' \wedge \tau'}$$

For Δ, Δ' non-linear environments such that $dom(\Delta) = dom(\Delta')$, we define $\Delta \leq \Delta'$ if for all variables in the domain of the environments the corresponding types in Δ and Δ' are $\leq$-related.
For any term M, two judgements assigning types to M are $\leq$-related if the linear environments are equal, while the non-linear environments and the assigned types are $\leq$-related.

▶ **Lemma 28.** *If the term M is typable, then there exists a principal judgement $\Gamma; \Delta \Vdash M : \tau$,* i.e. *a minimal judgement w.r.t. $\leq$, which is unique up-to α-renaming.*

Proof. By induction on derivations. In order to deal with the (app)-rule, we need to prove that, if $\Gamma; \Delta \Vdash M : \sigma \multimap \sigma'$, $\Gamma'; \Delta' \Vdash N : \tau$, and $U = MGU(\sigma, \tau)$, then either the principal type of M is a variable or there exist minimal judgements $\Gamma_1; \Delta_1 \Vdash M : \sigma_1 \multimap \sigma'_1$, $\Gamma'_1; \Delta'_1 \Vdash N : \tau_1$ such that $U' = MGU(\sigma_1, \tau_1)$. ◀

Here are the principal types of the combinators:

I	$\lambda x.x$	$\alpha \multimap \alpha$	D	$\lambda !x.x$	$!_\epsilon\alpha \multimap \alpha$
K	$\lambda xy.x$	$\alpha \multimap \beta \multimap \alpha$	δ	$\lambda !x.!!x$	$!_{<i,j>}\alpha \multimap !_i!_j\alpha$
B	$\lambda xyz.x(yz)$	$(\alpha \multimap \gamma) \multimap (\beta \multimap \alpha) \multimap \beta \multimap \gamma$	F	$\lambda !x!y.!(xy)$	$!_i(\alpha \multimap \beta) \multimap !_i\alpha \multimap !_i\beta$
C	$\lambda xyz.xzy$	$(\alpha \multimap \beta \multimap \gamma) \multimap \beta \multimap \alpha \multimap \gamma$	W	$\lambda x!y.x!y!y$	$(!_i\alpha \multimap !_j\beta \multimap \gamma) \multimap (!_{li}\alpha \wedge !_{rj}\beta) \multimap \gamma$

As we will see in Secion 6, the principal types of the combinators induce, via the transformation $\mathcal{I}$ of Definition 32, the corresponding partial involutions (see Proposition 11).

Another intriguing example of the *involutions-as-types* analogy is the following.

Let us consider the $CL^!$-terms $F(!I)$, $B(F!D)\delta$, and $BD\delta$. Despite having the same applicative behaviour on !-arguments, as can be easily seen by reducing them, these terms are interpreted by three different partial involutions in the combinatory algebra $\mathcal{P}$:

- $F(!I) \ : \ l\langle x, y\rangle \leftrightarrow r\langle x, y\rangle$,
- $B(F!D)\delta \ : \ l\langle\langle x, \epsilon\rangle, y\rangle \leftrightarrow r\langle x, y\rangle$,
- $BD\delta \ : \ l\langle\langle \epsilon, x\rangle, y\rangle \leftrightarrow r\langle x, y\rangle$.

Quite correctly, the three terms above turn out to have also different principal types (in what follows we denote, by abuse of notation, the λ!-terms corresponding to the $CL^!$-terms directly by the $CL^!$-terms themselves):

- $\Vdash F(!I) : !_i\alpha \multimap !_i\alpha$,
- $\Vdash B(F!D)\delta : !_{\langle i,\epsilon\rangle}\alpha \multimap !_i\alpha$,
- $\Vdash BD\delta : !_{\langle \epsilon,i\rangle}\alpha \multimap !_i\alpha$.

As we will see in Section 6, the above principal types exactly correspond to the expected partial involutions.

5.1 Relating the Principal Type System to the Type System

In the following, we study the relationships between the two typing systems. As expected, they are related via substitutions U. In order to state precisely the correspondence between the two intersection type systems, we need the following lemma, which can be proved by induction on derivations:

▶ **Lemma 29.** *If $\Gamma; \Delta \vdash M : \sigma$, then, for all substitutions U, $U(\Gamma); U(\Delta) \vdash M : U(\sigma)$.*

▶ **Theorem 30.** *For all $M \in \mathbf{\Lambda}^!$:*

(i) *if $\Gamma;\Delta \Vdash M : \sigma$, then, for all substitutions U, $U(\Gamma);U(\Delta) \vdash M : U(\sigma)$;*

(ii) *if $\Gamma;\Delta \vdash M : \sigma$, then there exist a derivation $\Gamma',\Delta' \Vdash M : \sigma'$ and a type substitution U such that $U(\Gamma') = \Gamma$, $U(\Delta') = \Delta$, $U(\sigma') = \sigma$.*

Proof. Both items can be proved by induction on derivations. Lemma 29 above is used to prove item (i) in the case of (app)-rule. ◀

As a consequence of the above theorem, subject reduction/conversion results analogous to those in Theorems 19, 20, 21 hold for principal types:

▶ **Theorem 31** (Subject Reduction/Conversion)**.**

(i) *If $\Gamma;\Delta \Vdash M : \tau$ is a principal judgement and $M \to_\beta M'$, then $\exists \Gamma',\Delta',\tau'.(\Gamma';\Delta' \Vdash M' : \tau' \;\&\; \Gamma' = \Gamma_{|FV(M')} \;\&\; \Delta' \preceq \Delta \;\&\; \tau' \preceq \tau)$.*

(ii) *If $\Gamma;\Delta \Vdash M : \tau$ is a principal judgement and $M =^L_\beta M'$, then $\exists \Gamma',\Delta'.(\Gamma';\Delta' \Vdash M' : \tau \;\&\; \Gamma'_{|FV(M)\cap FV(M')} = \Gamma_{|FV(M)\cap FV(M')} \;\&\; \Delta' \sim \Delta)$.*

(iii) *If $\Gamma;\Delta \Vdash M : \tau$ is a principal judgement, $M =^C_\beta M'$, M' is typable, then $\Gamma;\Delta \Vdash M' : \tau$.*

Proof. We proof item(i), the proof of the remaining items being similar. If $\Gamma;\Delta \Vdash M : \tau$, then by Theorem 30(i) $\Gamma;\Delta \vdash M : \tau$, and by Theorem 19, $\Gamma';\Delta' \vdash M' : \tau'$, with $\Gamma' = \Gamma_{|FV(M')}$, $\Delta' \preceq \Delta$, $\tau' \preceq \tau$. Then, by Theorem 30(ii), $\Gamma'';\Delta'' \Vdash M' : \tau''$, with $U(\Gamma'') = \Gamma'$, $U(\Delta'') = \Delta'$, $U(\tau'') = \tau'$, for some substitution U. Hence, by Theorem 30(i), $\Gamma'';\Delta'' \vdash M' : \tau''$. Then, since M is typable in $\vdash$, the converse implication in Theorem 19 holds, and we have $\Gamma''';\Delta''' \vdash M : \tau'''$, with $\Gamma'' = \Gamma'''_{|FV(M')}$, $\Delta'' \preceq \Delta'''$, $\tau'' \preceq \tau'''$. Then, by Theorem 30(ii), $\overline{\Gamma};\overline{\Delta} \Vdash M : \overline{\tau}$, with $U'(\overline{\Gamma}) = \Gamma'''$, $U'(\overline{\Delta}) = \Delta'''$, $U'(\overline{\tau}) = \tau'''$, for some substitution U'. Hence, by unicity of the principal judgement, $\overline{\Gamma} = \Gamma$, $\overline{\Delta} = \Delta$, $\overline{\tau} = \tau$. Finally, we are left to prove that $\Gamma'' = \Gamma_{|FV(M')}$, $\Delta'' \preceq \Delta$, $\tau'' \preceq \tau$. From $U(\Gamma'') = \Gamma' = \Gamma_{|FV(M')} = \overline{\Gamma}_{|FV(M')}$ and $U'(\overline{\Gamma}_{|FV(M')}) = \Gamma'''_{|FV(M')} = \Gamma''$ it follows $\Gamma'' = \Gamma_{|FV(M')}$. From $U(\Delta'') = \Delta' \preceq \Delta = \overline{\Delta}$ and $U'(\overline{\Delta}) = \Delta''' \succeq \Delta''$ it follows $\Delta'' \preceq \Delta$. Similarly, we get $\tau'' \preceq \tau$. ◀

6 Relating Principal Types and Partial Involutions

In this section, we state precisely the correspondence between principal type schemes and partial involutions, giving evidence to the *involutions-as-types* analogy. In particular, we provide procedures for building the partial involution corresponding to a type, and back.

The following algorithm, given a principal type scheme, produces the corresponding involution:

▶ **Definition 32.** *For α type variable and τ type, we define the judgement $\mathcal{I}(\alpha,\tau)$, which, if it terminates, gives a pair in the graph of the partial involution, if α occurs twice in τ, or an element of T_Σ, if α occurs once in τ:*

$$\begin{aligned}
\mathcal{I}(\alpha,\alpha) &= \alpha\\
\mathcal{I}(\alpha,\sigma(\alpha) \multimap \tau(\alpha)) &= l\mathcal{I}(\alpha,\sigma(\alpha)) \;\leftrightarrow\; r\mathcal{I}(\alpha,\tau(\alpha))\\
\mathcal{I}(\alpha,\sigma(\alpha) \multimap \tau) &= l\mathcal{I}(\alpha,\sigma(\alpha))\\
\mathcal{I}(\alpha,\sigma \multimap \tau(\alpha)) &= r\mathcal{I}(\alpha,\tau(\alpha))\\
\mathcal{I}(\alpha,\sigma(\alpha) \wedge \tau(\alpha)) &= \mathcal{I}(\sigma(\alpha)) \leftrightarrow \mathcal{I}(\tau(\alpha))\\
\mathcal{I}(\alpha,\sigma(\alpha) \wedge \tau) &= \mathcal{I}(\alpha,\sigma(\alpha))\\
\mathcal{I}(\alpha,\tau \wedge \sigma(\alpha)) &= \mathcal{I}(\alpha,\sigma(\alpha))\\
\mathcal{I}(\alpha,!_u\tau(\alpha)) &= \langle u,\mathcal{I}(\alpha,\tau(\alpha))\rangle
\end{aligned}$$

where, by abuse of notation, when r, l apply to a pair, we mean that they apply to the single components.

We define the partial involution induced by the type τ:

$$f_\tau = \{\mathcal{I}(\alpha, \tau) \mid \alpha \textit{ appears twice in } \tau\}.$$

▶ **Definition 33.** *Having selected a special type variable ω, we define a partial function $\mathcal{T}$ which, given a partial involution term $t \in T_\Sigma$, returns a type:*

$$\begin{aligned}
\mathcal{T}(\alpha) &= \alpha \\
\mathcal{T}(lt) &= \mathcal{T}(t) \multimap \omega \\
\mathcal{T}(rt) &= \omega \multimap \mathcal{T}(t) \\
\mathcal{T}(\langle t_1, t_2 \rangle) &= !_{t_1} \mathcal{T}(t_2)
\end{aligned}$$

On types we define a partial operation $\cup$ as follows:

$$\omega \cup \tau = \tau \cup \omega = \tau$$
$$(\sigma_1 \multimap \tau_1) \cup (\sigma_2 \multimap \tau_2) = (\sigma_1 \cup \sigma_2) \multimap (\tau_1 \cup \tau_2)$$
$$!_u \tau \cup !_v \sigma = U(!_u(\tau \cup \sigma)) \textit{ if } \exists U = MGU(u, v)$$
$$!_u \tau \cup (!_v \sigma_1 \wedge \sigma_2) = U(!_u(\tau \cup \sigma_1) \wedge \sigma_2) \textit{ if } \exists U = MGU(u, v)$$
$$!_u \tau \cup (!_{v_1} \sigma_1 \wedge \cdots \wedge !_{v_n} \sigma_n) = !_u \tau \wedge !_{v_1} \sigma_1 \wedge \cdots \wedge !_{v_n} \sigma_n \textit{ if } \forall i \,.\, \nexists U = MGU(v_i, u)$$

For each partial involution $\pi = \{t_1 \leftrightarrow t'_1, \ldots, t_n \leftrightarrow t'_n\}$, we define its associated type as: $\mathcal{T}(\pi) = \mathcal{T}(t_1) \cup (\mathcal{T}(t'_1) \cup (\mathcal{T}(t_2) \cup \ldots \cup (\mathcal{T}(t_n) \cup \mathcal{T}(t'_n) \ldots)$.

Finally, we can show that type unification corresponds to application of involutions:

▶ **Theorem 34.** *Let $\sigma \multimap \tau$, σ' be binary types such that $U = MGU(\sigma, \sigma')$. Then $\mathcal{I}(\sigma \multimap \tau) \cdot \mathcal{I}(\sigma') = \mathcal{I}(U(\tau))$.*

Proof. (Sketch) One can prove that, under the hypothesis that $MGU(\sigma, \sigma')$ exists, in evaluating $\mathcal{I}(\sigma \multimap \tau) \cdot \mathcal{I}(\sigma')$ one constructs, in a series of steps, the unifier between the types σ and σ', and the final step of a linear application interaction corresponds to the application of the unifier to τ. ◀

As a consequence of the above theorem, principal types of λ!-terms correspond to partial involutions interpreting the terms in the combinatory algebra of partial involutions:

▶ **Theorem 35.** *Given a closed term of $\mathbf{CL}^!$, say M, such that $(M)_{\lambda^!}$ is typable, the partial involution interpreting M, namely $[\![M]\!]_{\mathcal{P}}$, can be read off from the principal type scheme of $(M)_{\lambda^!}$, i.e. $\Vdash (M)_{\lambda^!} : \tau$ if and only if $[\![M]\!]_{\mathcal{P}} = f_\tau$.*

Proof. The thesis follows from Theorem 34 and from the fact that λ!-terms corresponding to combinatory constants receive the principal types inducing the partial involutions interpreting the combinatory constants. ◀

7 Final Remarks and Directions for Future Work

In this paper, we have analysed from the point of view of the model theory of λ-calculus the affine combinatory algebra of partial involutions, $\mathcal{P}$, introduced in [1]. The key insight which has allowed us to analyze the fine structure of the partial involutions interpreting

combinators has been what we termed the *involutions-as-principal types/application-as-resolution* analogy, introduced in [8, 9], which highlights a form of structural duality between involutions and principal types, w.r.t. a suitable intersection type discipline. We feel that it offers a new perspective on Girards's Geometry of Interaction and especially on how its reversible dynamics arises. Our next step is to explore how to apply this paradigm to other instances of Game Semantics and GoI situations.

There are also many interesting lines of future work that remain to be addressed as far as partial involutions are concerned. In particular, the type assignment systems can be refined or extended in several directions.

- First of all, both type systems introduced in this paper could be further fine-tuned in order to capture even more smoothly the partial involutions corresponding to the constants of affine combinatory algebras. *E.g.* the functorial nature of F could be taken as a rule.
- Our type system is able to type only normalizable λ-terms. By introducing an extra type constant ω representing an undefined type, it is possible to assign types to general λ-terms. In this case, λ-terms generating infinitary Böhm trees will be characterized by a set of principal types, each type defining a finite approximation of the term.
- The present principal type system is not completely "deterministic", *i.e.* in general a set of types can be assigned to a λ-term, but only one is principal. We aim at developing an alternative type assignment system where only principal types are derivable.
- The type assignment systems defined in this paper induce combinatory algebras but fail to be λ-algebras. It would be interesting to explore suitable quotients inducing full λ-algebras.
- Similarly, the combinatory algebra of partial involutions also fails to be a λ-algebra, and therefore it would be worth to investigate how to quotient it to get a λ-algebra.
- A further interesting problem to address is to characterize the fine theory of $\mathcal{P}$. This should be done by proving a suitable Approximation Theorem, relying on a complexity measure on involutions, induced by a complexity measure on words in T_Σ.
- Building on the results of this paper, we should be able to provide an answer to the open problem raised in [1] of characterising the partial involutions which arise as denotations of combinators, extending the solution given in [9] for the purely affine fragment.
- Comparison with alternate λ-calculi for describing reversible computations, *e.g.* [11], or other typing systems inspired to Linear Logic, *e.g.* [10], should be carried out.

References

1. Samson Abramsky. A structural approach to reversible computation. *Theoretical Computer Science*, 347(3):441–464, 2005. `doi:10.1016/j.tcs.2005.07.002`.
2. Samson Abramsky, Esfandiar Haghverdi, and Philip Scott. Geometry of Interaction and linear combinatory algebras. *Mathematical Structures in Computer Science*, 12(5):625–665, 2002. `doi:10.1017/S0960129502003730`.
3. Samson Abramsky and Radha Jagadeesan. Games and full completeness for multiplicative linear logic. *Journal of Symbolic Logic*, 59(2):543–574, 1994. `doi:10.2307/2275407`.
4. Samson Abramsky and Marina Lenisa. Linear realizability and full completeness for typed lambda-calculi. *Annals of Pure and Applied Logic*, 134(2):122–168, 2005. `doi:10.1016/j.apal.2004.08.003`.
5. Joe Armstrong. *Making reliable distributed systems in the presence of software errors*. PhD thesis, KTH, Microelectronics and Information Technology, IMIT, 2003. NR 20140805.
6. Henk Barendregt, Mario Coppo, and Mariangiola 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`.

7 HP Barendregt. *The Lambda Calculus. Its Syntax and Semantics*. North-Holland, Amsterdam, 1984. (revised edition).

8 Alberto Ciaffaglione, Pietro Di Gianantonio, Furio Honsell, Marina Lenisa, and Ivan Scagnetto. Reversible Computation and Principal Types in λ!-calculus. *The Bulletin of Symbolic Logic*, 2018.

9 Alberto Ciaffaglione, Furio Honsell, Marina Lenisa, and Ivan Scagnetto. The involutions-as-principal types/application-as-unification Analogy. In Gilles Barthe, Geoff Sutcliffe, and Margus Veanes, editors, *LPAR*, volume 57 of *EPiC Series in Computing*, pages 254–270. EasyChair, 2018. URL: `http://dblp.uni-trier.de/db/conf/lpar/lpar2018.html#CiaffaglioneHLS18`.

10 Ugo Dal Lago and Barbara Petit. The geometry of types. In *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL'13, Proceedings*, pages 167–178. ACM, 2013.

11 Alessandra Di Pierro, Chris Hankin, and Herbert Wiklicky. Reversible combinatory logic. *Mathematical Structures in Computer Science*, 16(4):621–637, 2006. `doi:10.1017/S0960129506005391`.

12 Erlang official website. Last access: 19/01/2018. URL: `http://www.erlang.org`.

13 Pietro Di Gianantonio, Furio Honsell, and Marina Lenisa. A type assignment system for game semantics. *Theoretical Computer Science*, 398(1):150–169, 2008. Calculi, Types and Applications: Essays in honour of M. Coppo, M. Dezani-Ciancaglini and S. Ronchi Della Rocca. `doi:10.1016/j.tcs.2008.01.023`.

14 Pietro Di Gianantonio and Marina Lenisa. Innocent Game Semantics via Intersection Type Assignment Systems. In Simona Ronchi Della Rocca, editor, *Computer Science Logic 2013 (CSL 2013)*, volume 23 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 231–247, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/LIPIcs.CSL.2013.231`.

15 Jean-Yves Girard. Geometry of interaction 2: Deadlock-free algorithms. In Per Martin-Löf and Grigori Mints, editors, *COLOG-88*, pages 76–93, Berlin, Heidelberg, 1990. Springer Berlin Heidelberg.

16 Jean-Yves Girard. Geometry of interaction III: accommodating the additives. *London Mathematical Society Lecture Note Series*, pages 329–389, 1995.

17 Esfandiar Haghverdi. *A categorical approach to linear logic, geometry of proofs and full completeness*. University of Ottawa (Canada), 2000.

18 Alex Simpson. Reduction in a Linear Lambda-Calculus with Applications to Operational Semantics. In Jürgen Giesl, editor, *Term Rewriting and Applications*, pages 219–234, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

19 Web Appendix with Erlang code. URL: `http://www.dimi.uniud.it/scagnett/pubs/automata-erlang.pdf`.

Template Games, Simple Games, and Day Convolution

Clovis Eberhart
National Institute of Informatics, Tokyo, Japan
http://group-mmm.org/~eberhart/

Tom Hirschowitz
Univ. Grenoble Alpes, Univ. Savoie Mont Blanc, CNRS, LAMA, 73000, Chambéry, France
https://www.lama.univ-savoie.fr/pagesmembres/hirschowitz

Alexis Laouar
Univ. Grenoble Alpes, Univ. Savoie Mont Blanc, CNRS, LAMA, 73000, Chambéry, France

Abstract

Template games [14] unify various approaches to game semantics, by exhibiting them as instances of a double-categorical variant of the slice construction. However, in the particular case of simple games [9, 12], template games do not quite yield the standard (bi)category. We refine the construction using factorisation systems, obtaining as an instance a slight generalisation of simple games and strategies. This proves that template games have the descriptive power to capture combinatorial constraints defining well-known classes of games. Another instance is Day's *convolution* monoidal structure on the category of presheaves over a strict monoidal category [2], which answers a question raised in [3].

2012 ACM Subject Classification Theory of computation → Denotational semantics

Keywords and phrases Game semantics, Day convolution, Categorical semantics

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.16

Funding *Clovis Eberhart*: ERATO HASUO Metamathematics for Systems Design Project (No. JPMJER1603, http://dx.doi.org/10.13039/501100009024), JST.

1 Introduction

Game semantics has provided adequate models for a variety of programming languages [11], in which types are interpreted as two-player games and programs as strategies. Most game models follow a common pattern. Typically, a function $A \to B$ is interpreted as a strategy on a compound game made of A and B, where the program plays as *Proponent* (P) on B and as *Opponent* (O) on A. Another common feature is composition of strategies, which takes strategies $\sigma\colon A \to B$ and $\tau\colon B \to C$, and returns a strategy $\tau \circ \sigma\colon A \to C$ by letting σ and τ interact on B until one of them produces a move in A or C.

Although widely acknowledged, this strong commonality is also recognised as poorly understood, particularly in the presence of *innocence*, a constraint on strategies that restricts them to purely functional behaviour. This has prompted a number of attempts at clarifying the situation [10, 9, 4]. Recently, Melliès [14] proposed a novel explanation, of unprecedented simplicity, named *template games*. It is based upon a purely categorical construction, essentially taking the slice of a weak double category over an internal monad, the *template*. This produces a new weak double category, in which composition of strategies occurs as so-called horizontal composition. In order to illustrate the construction, Melliès applies it to

© Clovis Eberhart, Tom Hirschowitz, and Alexis Laouar;
licensed under Creative Commons License CC-BY
4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019).
Editor: Herman Geuvers; Article No. 16; pp. 16:1–16:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

three different templates to obtain three models, respectively related to simple games [12], concurrent games [15], and the relational model. However, the first two differ significantly from their more standard counterparts.

This raises the question of whether template games only produce new game models, or whether they also cover standard models. In this paper, we show that, up to a slight refinement, they do cover simple games. More precisely, we (1) modify the original template for simple games (Lemma 64), and (2) enrich the general construction with a factorisation system [1] (Theorem 61), obtaining a slight generalisation of standard simple games as an instance (Corollary 68). More precisely, we obtain a variant in which games may have several 'initial positions', before the game even starts, and similarly strategies may have several 'initial states' over each of these initial positions. We thus easily characterise standard simple games and strategies as so-called *definite* template games and strategies.

One motivation for simple and abstract constructions like template games is to find new connections with other settings. Our refined construction yields one such connection: we show that the Day *convolution* product [2] arises as an instance of our refined framework, though only in the restricted case of *strict* monoidal categories (Theorem 73). The convolution product, which arose in algebraic topology, extends the monoidal structure of a given category $\mathbb{C}$ to the category $\widehat{\mathbb{C}}$ of *presheaves* on $\mathbb{C}$, i.e., contravariant functors $\mathbb{C}^{op} \to \mathsf{Set}$. This makes formal the similarity, noted in [3, §6.5], between convolution and composition of strategies, by showing that both are instances of the same construction.

Related work

Beyond [14] and the related [10], Garner and Shulman [7] prove results related to our Theorems 52 and 61. The common ground for comparison is the restriction of Theorem 52 to weak double categories with a trivial vertical category, i.e., monoidal categories. Their Theorem 14.2 is a generalisation in another direction, namely that of monoidal bicategories, and their Theorem 14.5 could in particular accomodate various sorts of bicategorical factorisation systems.

Terminology

Although template games and strategies are an abstract construction, we often abuse the term to denote the particular instance on the template for simple games.

Plan

We follow the standard construction layers of game models: games, strategies, and composition of strategies. In Section 2, we analyse the differences between template and simple games, and describe our refinement of the former, which allows us to bridge the gap. In Section 3, we do the same at the level of strategies $A \to B$, for fixed games A and B. In Section 4, we recall the abstract construction of template games. In Section 5, we introduce our refined construction, and illustrate it on the promised instances.

2 Template games vs. simple games

In this section, we first recall template games, and then analyse the discrepancies with simple games. Finally, we introduce our solution to bridge the gap between them.

Template games are based on the following simple category.

▶ **Definition 1.** *Let* $\pm_v$ *denote the category freely generated by the graph* $O \rightleftarrows P$.

Thus, objects are just O and P, which stand for *Opponent* and *Proponent*, as in most game models, and morphisms just count the number of (alternating) moves between them.

▶ **Definition 2.** *A* template game *is a category* A*, equipped with a functor* $p\colon A \to \pm_v$*.*

Intuitively, objects of A are positions, or plays, in the game, and p gives their polarity: by convention, O is to play in positions mapped to O, while P is to play in others.

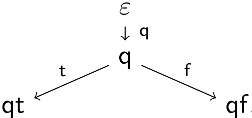
Simple games clearly fit into this framework.

▶ **Definition 3.** *A* simple game A *is a rooted tree.*

The intuition is the same as for template games, and indeed:

▶ **Proposition 4.** *Any simple game* A*, viewed as a poset, hence as a category, forms a template game* $p_A\colon A \to \pm_v$*, where* p_A *maps the root to* O*, its children to* P*, and so on.*

▶ **Example 5.** The simple game $\mathbb{B}$ for booleans is the tree



where we have labelled edges with moves, and each node with the corresponding play, i.e., the sequence of moves needed to reach it from the root. The first move, q, represents O asking the value of the boolean, and the other moves represent the possible answers: true and false. The functor $p_{\mathbb{B}}\colon \mathbb{B} \to \pm_v$ maps q to P and all other objects to O.

Discrepancies between template and simple games

Template games are significantly more general than simple games. To start with, simple games have an empty, initial play, which template games need not. Furthermore, this initial play is mapped to O by the functor to $\pm_v$.

▶ **Lemma 6.** *For any simple game* A*, the category* A *has an initial object, mapped to* O *by the functor* $A \to \pm_v$*.*

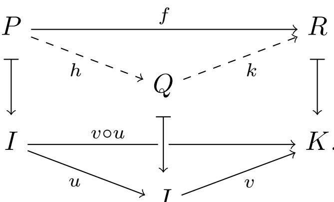
Let us exhibit some counterexamples among general template games.

▶ **Example 7.** The template game $\mathbb{1} \to \pm_v$ picking up P is not equivalent to any simple game in the slice 2-category $\mathsf{Cat}/\pm_v$.

▶ **Example 8.** Consider the template game consisting of $\mathbb{Z}$, the poset of integers, and the functor $\mathbb{Z} \to \pm_v$ that maps $2n$ to O and $2n+1$ to P. The category $\mathbb{Z}$ has no initial object, hence this template game is not equivalent to any simple game in $\mathsf{Cat}/\pm_v$.

Another discrepancy has to do with decomposing plays into moves.

▶ **Definition 9.** *A functor $F\colon \mathbb{E} \to \mathbb{B}$ is a* discrete Conduché fibration *iff for any $I \xrightarrow{u} J \xrightarrow{v} K$ in $\mathbb{B}$ and $P \xrightarrow{f} R$ in $\mathbb{E}$ mapped to $v \circ u$, f uniquely factors as $k \circ h$ with $F(h) = u$ and $F(k) = v$:*



▶ **Lemma 10.** *For any simple game A, the functor $p_A\colon A \to \pm_v$ is a discrete Conduché fibration.*

Proof. By construction, a morphism in A over a path of length n has the form $p \to pm_1 \dots m_n$ for some play p and moves $m_1, \dots, m_n$, hence decomposes as needed. ◀

Unlike simple games, not all template games are Conduché fibrations. Indeed, they may feature *atomic* sequences of moves, i.e., morphisms that are mapped to non-basic morphisms in $\pm_v$ and yet are 'indecomposable'.

▶ **Example 11.** Consider the ordinal $\mathbb{2} = \{0 \leq 1\}$ viewed as a category, and the functor $\mathbb{2} \to \pm_v$ mapping $0 \leq 1$ to the path $O \to P \to O$. This is clearly not a Conduché fibration, hence is not equivalent to any simple game.

Refining template games

As announced in §1, our solution to bridge the gap between template and simple games is twofold: (1) we use a different template, $\mathbb{T}$, and (2) we introduce a factorisation system into the picture – concretely, we restrict attention to functors $A \to \mathbb{T}$ which are discrete fibrations. Let us start with the new template.

▶ **Definition 12.** *Let $\mathbb{T}_v = \omega$ denote the poset of natural numbers, seen as a category.*

▶ Remark 13. $\mathbb{T}_v$ is isomorphic to the coslice category $O/\pm_v$.

Defining games to be functors to $\mathbb{T}_v$ intuitively goes in the right direction, but does not quite solve any of our two problems.

▶ **Example 14.** Consider the functor $D\colon \omega \to \omega = \mathbb{T}_v$ defined by $D(n) = 2n + 1$. It exhibits both problems at once: it does not preserve the initial object, and, e.g., $0 \leq 1$ does not admit any decomposition along the decomposition $1 \leq 2 \leq 3$, a.k.a. $P \to O \to P$, of its image.

However, if we restrict attention to a certain kind of functors $A \to \mathbb{T}_v$, we solve both problems at once – almost. The relevant constraint on functors is generally stronger than being a Conduché fibration, but becomes equivalent when both categories have an initial object which is preserved by the functor.

▶ **Definition 15.** *A functor $F\colon \mathbb{E} \to \mathbb{B}$ is a* discrete fibration *when for any $E \in \mathbb{E}$ and $u\colon B \to F(E)$, there is a unique $f\colon E' \to E$ such that $F(f) = u$, as in*

$$\begin{array}{ccc} E' & \overset{f}{\dashrightarrow} & E \\ \downarrow & & \downarrow \\ B & \xrightarrow[u]{} & F(E). \end{array}$$

Let $\mathsf{DFib}(\mathbb{C})$ *denote the full subcategory of* $\mathsf{Cat}/\mathbb{C}$ *spanning discrete fibrations.*

▶ **Definition 16.** *A* refined template game *is a discrete fibration to* $\mathbb{T}_v$*.*

Of course, any simple game A yields a discrete fibration $p_A \colon A \to \mathbb{T}_v$, and we have:

▶ **Proposition 17.** *Any refined template game in* $\mathsf{DFib}(\mathbb{T}_v)$ *is isomorphic to* p_A*, for some simple game* A*, iff it is* definite*, i.e., its fibre over* 0 *is a singleton.*

3 Template strategies vs. simple strategies

Let us now consider strategies. As for games, we start with Melliès's notion, to emphasise its simplicity. Template strategies are based on the following simple category:

▶ **Definition 18.** *Let* $\pm_V$ *denote the category freely generated by the graph*

$$OO \rightleftarrows OP \rightleftarrows PP.$$

This is essentially the well-known state diagram for strategies in a simple arrow game $A \to B$, extended to a category. We need a little lemma before defining strategies.

▶ **Lemma 19.** *The left and right projections give rise to functors* $s, t \colon \pm_V \to \pm_v$*, with* $s(XY) = X$ *and* $t(XY) = Y$*.*

▶ **Definition 20.** *A* template strategy *from* $p \colon A \to \pm_v$ *to* $q \colon B \to \pm_v$ *is a tuple* (S, s', t', r) *making the following diagram commute.*

$$\begin{array}{ccccc} A & \xleftarrow{s'} & S & \xrightarrow{t'} & B \\ {\scriptstyle p}\downarrow & & \downarrow{\scriptstyle r} & & \downarrow{\scriptstyle q} \\ \pm_v & \xleftarrow[s]{} & \pm_V & \xrightarrow[t]{} & \pm_v \end{array} \qquad (1)$$

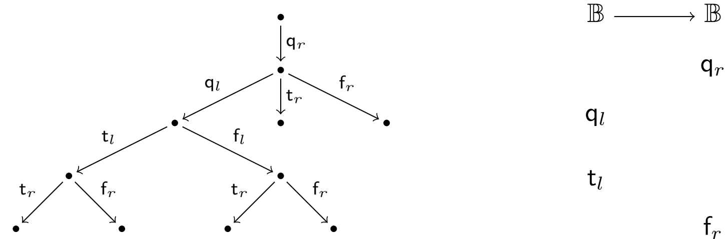
Let us now show that simple strategies give rise to template strategies. Simple strategies are rather subtle – which was one of the main motivations for template games in the first place! – so this is a bit technical. We first consider *boolean* simple strategies, which are easier, and then move on to general ones.

Boolean simple strategies

First, we need to define the *arrow* game $A \to B$, for any two simple games A and B. The following definition is slightly vague: we refer to [12] for a fully rigorous one.

▶ **Definition 21.** *Given two simple games* A *and* B*,* $A \to B$ *interleaves moves from* A *and* B *according to the following rules: (1) the polarity of moves in* A *is inverted, (2)* O *starts in* B*, and (3) only* P *gets to switch sides.*

▶ **Example 22.** The game $\mathbb{B} \to \mathbb{B}$ is depicted below left, with an example play on the right.



▶ **Definition 23.** *A* boolean simple strategy $A \to B$ *is a prefix-closed set of non-empty, even-length plays, called* accepted*, in* $A \to B$*.*

▶ **Example 24.** The set of non-empty, even-length prefixes of the play in Example 22 (i.e., the set $\{\mathsf{q}_r\mathsf{q}_l, \mathsf{q}_r\mathsf{q}_l\mathsf{t}_l\mathsf{f}_r\}$) forms a boolean simple strategy.

There are in fact several ways in which boolean simple strategies give rise to template strategies. We here present the relevant one in terms of semantics.

▶ **Definition 25.** *The template strategy associated to a boolean simple strategy* $\sigma \colon A \to B$ *is* $(\mathbb{E}(\sigma), s', t', r)$*, where* $\mathbb{E}(\sigma)$ *is the poset of all prefixes of accepted plays, plus all extensions* sm *of prefixes* s *of even length, and* r*,* s'*, and* t' *are the obvious projections.*

▶ **Example 26.** Consider the strategy of Example 24. The functor $\mathbb{E}(\sigma) \to \pm_V$ can be represented as

$$OO \xrightarrow{\mathsf{q}_r} OP \xrightarrow{\mathsf{q}_l} PP \begin{array}{l} \xrightarrow{\mathsf{t}_l} OP \xrightarrow{\mathsf{f}_l} OO \\ \xrightarrow{\mathsf{f}_l} OP, \end{array} \tag{2}$$

with $s', t' \colon \mathbb{E}(\sigma) \to \mathbb{B}$ displayed on the left and right in Figure 1 (with polarity indications).

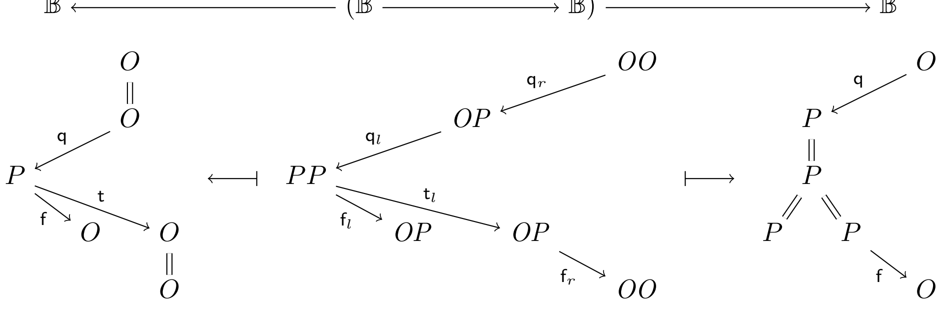


Figure 1 Template strategy associated to the simple strategy of Example 24.

General simple strategies

Let us now consider the more general, non-deterministic (i.e., non-boolean), but still standard, notion of simple strategy [12]. Let us start from an alternative presentation of boolean strategies.

▶ **Definition 27.** *Let $(A \to B)^{P*}$ denote the full subcategory of $(A \to B)$ spanning non-empty, even-length plays, and $\mathsf{i}_{P*} \colon (A \to B)^{P*} \hookrightarrow (A \to B)$ the inclusion functor.*

▶ **Proposition 28.** *Boolean simple strategies $A \to B$ are equivalent to functors $((A \to B)^{P*})^{op} \to \mathbb{2}$.*

The idea is that a strategy σ accepts a given play p iff the corresponding functor maps it to $1 \in \mathbb{2}$, observing that functoriality ensures prefix-closedness.

General simple strategies are obtained by generalising from $\mathbb{2}$ to Set:

▶ **Definition 29.** *Let the category of simple strategies $A \to B$ be $\mathcal{S}(A, B) := \widehat{(A \to B)^{P*}}$.*

▶ Remark 30. As noted in [12], this is equivalent to the standard definition as a slice category. Indeed, letting ω^{P*} denote the full subcategory of ω on positive, even ordinals, $p_{A \to B}$ restricts to a discrete fibration $(A \to B)^{P*} \to \omega^{P*}$. By the well-known equivalence $\partial^\star \colon \mathsf{DFib}(\mathbb{C}) \to \widehat{\mathbb{C}} :\mathrm{el}$ between presheaves and discrete fibrations (recalled as Lemma 32 below), and, for any $U \in \widehat{\mathbb{C}}$, the further equivalence $\widehat{\mathbb{C}}/U \simeq \widehat{\mathrm{el}(U)}$, we obtain

$$\mathcal{S}(A, B) = \widehat{(A \to B)^{P*}} \simeq \widehat{\omega^{P*}}/\partial^\star((A \to B)^{P*}).$$

The latter slice category is precisely the standard definition.

Of course, boolean simple strategies embed into general ones by postcomposition with the embedding $\mathbb{2} \hookrightarrow \mathsf{Set}$ determined by $0 \mapsto \varnothing$ and $1 \mapsto 1$.

From simple strategies to template strategies

Let us now informally describe how simple strategies $\sigma \in \widehat{(A \to B)^{P*}}$ give rise to template strategies, for which the following well-known result is the basis.

▶ **Definition 31.** *For any small category $\mathbb{C}$, let $X \in \widehat{\mathbb{C}}$. The* category of elements $\mathrm{el}(X)$ *of X has as objects pairs (c, x) with $x \in X(c)$, and as morphisms $(c, x) \to (c', x')$ all morphisms $f \colon c \to c'$ such that $X(f)(x') = x$.*

▶ **Lemma 32.** *For any small category $\mathbb{C}$ and $X \in \widehat{\mathbb{C}}$, the projection functor $p_X \colon \mathrm{el}(X) \to \mathbb{C}$ is a dicrete fibration, and the category of elements construction extends to an adjoint equivalence of categories $\mathrm{el} \colon \widehat{\mathbb{C}} \to \mathsf{DFib}(\mathbb{C})$. Let $\partial^\star$ denote the weak inverse to* el.

▶ **Example 33.** One possible definition of forests is as presheaves over ω. Indeed, any $T \in \widehat{\omega}$ models the forest whose nodes of depth n are $T(n)$, and whose parent relation is given by $T(n \leq n+1) \colon T(n+1) \to T(n)$. The category $\mathrm{el}(T)$ is just the same forest viewed as a poset, or rather a category, and the projection functor $p_T \colon \mathrm{el}(T) \to \omega$ maps vertices to their depths. Trees are those $T \in \widehat{\omega}$ such that $T(0)$ is a singleton.

Returning to simple and template strategies, we have a commuting diagram of functors

$$\begin{array}{ccccc} A & \longleftarrow & (A \to B) & \longrightarrow & B \\ {\scriptstyle p_A}\downarrow & & \downarrow & & \downarrow{\scriptstyle p_B} \\ \pm_v & \xleftarrow[s]{} & \pm_V & \xrightarrow[t]{} & \pm_v, \end{array} \tag{3}$$

so a first candidate follows by composition with the functor

$$\mathrm{el}(\sigma) \to (A \to B)^{P*} \xhookrightarrow{\mathsf{i}_{P*}} (A \to B).$$

However, the fibres of this functor over plays of odd length are all empty. We thus need to insert additional objects over odd-length plays whose immediate prefix is accepted, which will at last induce the desired template strategy by composition with (3). We do this using right Kan extension:

▶ **Definition 34.** *Let* $\overline{\sigma} \in \widehat{A \to B} := \prod_{\mathfrak{i}_{P*}} \sigma$*, i.e., the right Kan extension of* σ *along* $\mathfrak{i}_{P*}^{op}$.

This does yield the desired behaviour, by a direct application of the well-known end formula for right Kan extension:

▶ **Lemma 35.** *We have:*

- $\overline{\sigma}(\varepsilon) = 1$*;*
- *for all odd-length plays* sm*,* $\overline{\sigma}(sm) = \sigma(s)$*;*
- *for all non-empty, even-length plays* s*,* $\overline{\sigma}(s) = \sigma(s)$*.*

The discrete fibration corresponding to $\overline{\sigma}$, $\mathrm{el}(\overline{\sigma})$, thus yields the desired template strategy.

However, we obtain the following result, which shows that even the more general simple strategies do not cover all template strategies.

▶ **Lemma 36.** *Both functors* $\mathrm{el}(\overline{\sigma}) \to (A \to B) \to \pm_V$ *are discrete Conduché fibrations. Furthermore, the former functor* $\mathrm{el}(\overline{\sigma}) \to (A \to B)$ *is* receptive*, in the sense that for all even-length plays* s *with immediate extensions* sm*, and all* $(s, x) \in \mathrm{el}(\overline{\sigma})$*, there exists a unique* $(sm, y) \in \mathrm{el}(\overline{\sigma})$ *and morphism* $(s, x) \to (sm, y)$ *mapped to* $s \to sm$ *by the projection.*

Proof. An easy computation using the characterisation of right Kan extensions as ends. ◀

Refining template strategies

As we did for games, let us now analyse and resolve the discrepancies. For compatibility with what we did for games, we take the coslice $\pm_V$ under OO and restrict to discrete fibrations $S \to OO/\pm_V$. Analogously to Lemma 19, we have functors $O/\pm_v \xleftarrow{s} OO/\pm_V \xrightarrow{t} O/\pm_v$.

▶ **Definition 37.** *Recalling Remark 13,* naively refined template strategies *are just as template strategies in Definition 20, but over* $O/\pm_v \xleftarrow{s} OO/\pm_V \xrightarrow{t} O/\pm_v$*, and with* r *a discrete fibration.*

We obtain a first improvement:

▶ **Lemma 38.** *The game* $A \to B$ *is a limit*

$$\begin{array}{ccccc} A & \longleftarrow & (A \to B) & \longrightarrow & B \\ {\scriptstyle p_A}\downarrow & & \downarrow & & \downarrow{\scriptstyle p_B} \\ O/\pm_v & \xleftarrow[s]{} & OO/\pm_V & \xrightarrow[t]{} & O/\pm_v, \end{array} \tag{4}$$

and naively refined template strategies are equivalent to presheaves over $A \to B$*.*

Proof. The first statement essentially says that plays in $A \to B$ are uniquely determined by their projections and the interleaving schedule, which is clear. For the second statement, $(A \to B) \to OO/\pm_V$ is a discrete fibration, so naively refined template strategies are in one-to-one correspondence with discrete fibrations over $A \to B$, hence with presheaves. ◀

In order to precisely capture simple strategies, it remains to account for receptiveness. For this, our solution is to further restrict the template:

▶ **Definition 39.** *Let $\mathbb{T}_V$ denote the full subcategory of $OO/\pm_V$ spanning even-length schedules.*

▶ Remark 40. It is tempting to further restrict to non-empty, even-length schedules. However, the obtained span does not form a monad, hence the general template games construction does not apply.

▶ **Definition 41.** Refined template strategies *are defined just as template strategies in Definition 20, but over $\mathbb{T}_v \xleftarrow{s} \mathbb{T}_V \xrightarrow{t} \mathbb{T}_v$, and with r a discrete fibration.*

We obtain a restriction of Lemma 38 to the relevant plays:

▶ **Lemma 42.** *The full subcategory $(A \to B)^P \hookrightarrow (A \to B)$ spanning even-length plays is a limit*

$$\begin{array}{ccccc} A & \longleftarrow & (A \to B)^P & \longrightarrow & B \\ {\scriptstyle p_A}\downarrow & & \downarrow & & \downarrow{\scriptstyle p_B} \\ \mathbb{T}_v & \xleftarrow[s]{} & \mathbb{T}_V & \xrightarrow[t]{} & \mathbb{T}_v, \end{array} \tag{5}$$

and refined template strategies $A \to B$ are equivalent to presheaves over $(A \to B)^P$.

Proof. By repeated application of stability of discrete fibrations under pullback, $(A \to B)^P \to \mathbb{T}_V$ is a discrete fibration. Hence, in a diagram like (1) (with $\mathbb{T}$ instead of $\pm$), assuming $p, q \in \mathsf{DFib}$, r is a discrete fibration iff the induced morphism to $(A \to B)^P$ is. The result then follows from Lemma 32. ◀

▶ **Corollary 43.** *There is a full, reflective embedding from simple strategies $\mathcal{S}(A \to B)$ to refined template strategies, whose essential image consists of* definite *refined template strategies, i.e., those whose associated presheaf $X \in \widehat{(A \to B)^P}$ is such that $X(\varepsilon) = 1$.*

Proof. The embedding $(A \to B)^{P*} \hookrightarrow (A \to B)^P$ being full, right Kan extension along its opposite defines a full, reflective embedding $\widehat{(A \to B)^{P*}} \hookrightarrow \widehat{(A \to B)^P}$, which returns definite refined template strategies by the standard end formula. ◀

Summary

Until now, we have refined template games and strategies, first by replacing the original template $\pm$ by our $\mathbb{T}$, and second by restricting template games and strategies to be discrete fibrations. We have then identified simple games as definite template games, and constructed a full, reflective embedding from simple strategies $A \to B$ to refined template strategies $p_A \to p_B$, with essential image the definite ones. What remains to be seen is whether we can refine Melliès's double-categorical construction accordingly.

4 Template games

In this section, we review Melliès's double-categorical variant of the slice construction. We then apply it to deduce that the new template $\mathbb{T}$ yields a weak double category as desired. In the next section, we refine the construction using a factorisation system, which allows us to account for restriction to discrete fibrations.

4.1 Double categories

The key point is that the template $\pm$ forms a *double category* [5], in a way that describes the scheduling of composition of strategies. So let us first briefly review double categories.

A double category $\mathcal{C}$ essentially consists of a *horizontal* category $\mathcal{C}_h$ and a *vertical* one $\mathcal{C}_v$ sharing the same object set, together with a set of *cells* as in

$$\begin{array}{ccc} A & \xrightarrow{\;f\;\bullet} & B \\ {\scriptstyle u}\downarrow & \Downarrow{\scriptstyle \alpha} & \downarrow{\scriptstyle v} \\ C & \xrightarrow[\;g\;]{\bullet} & D, \end{array}$$

where A, B, C, and D are objects, f and g are morphisms in $\mathcal{C}_h$, and u and v are morphisms in $\mathcal{C}_v$. In order to distinghish notationally between horizontal and vertical morphisms, we mark horizontal ones with a bullet. Cells are furthermore equipped with composition and identities in both directions. E.g., to any given cells α and β with compatible vertical border is assigned a composite cell $\beta \bullet \alpha$, as below left. Similarly, we have horizontal identities $id^{\bullet}_p$ as below right.

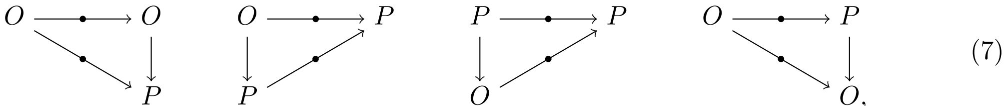
$$\begin{array}{ccccc} A & \xrightarrow{S} & B & \xrightarrow{S'} & E \\ {\scriptstyle p}\downarrow & \Downarrow{\scriptstyle \alpha} & \downarrow{\scriptstyle q} & \Downarrow{\scriptstyle \beta} & \downarrow{\scriptstyle r} \\ C & \xrightarrow[T]{} & D & \xrightarrow[T']{} & F \end{array} \;\mapsto\; \begin{array}{ccc} A & \xrightarrow{S' \bullet S} & E \\ {\scriptstyle p}\downarrow & \Downarrow{\scriptstyle \beta \bullet \alpha} & \downarrow{\scriptstyle r} \\ C & \xrightarrow[T' \bullet T]{} & F \end{array} \qquad \begin{array}{ccc} A & \xrightarrow{id^{\bullet}_A} & A \\ {\scriptstyle p}\downarrow & \Downarrow{\scriptstyle id^{\bullet}_p} & \downarrow{\scriptstyle p} \\ B & \xrightarrow[id^{\bullet}_B]{} & B \end{array} \tag{6}$$

Both notions of composition are required to be associative and the corresponding identities unital. Thus, e.g., (6) defines a *horizontal cell category* $\mathcal{C}_H$. Similarly, there is a *vertical cell category* $\mathcal{C}_V$. Finally, the *interchange law* requires the two different ways of parsing any compatible pasting as below to agree, i.e., $(\delta \circ \gamma) \bullet (\beta \circ \alpha) = (\delta \bullet \beta) \circ (\gamma \bullet \alpha)$:

$$\begin{array}{ccccc} A & \xrightarrow{\bullet} & B & \xrightarrow{\bullet} & C \\ \downarrow & \Downarrow{\scriptstyle \alpha} & \downarrow & \Downarrow{\scriptstyle \gamma} & \downarrow \\ D & \xrightarrow{\bullet} & E & \xrightarrow{\bullet} & F \\ \downarrow & \Downarrow{\scriptstyle \beta} & \downarrow & \Downarrow{\scriptstyle \delta} & \downarrow \\ G & \xrightarrow{\bullet} & H & \xrightarrow{\bullet} & I. \end{array}$$

4.2 The template $\pm$ as a double category

Let us now show that the template $\pm$ forms a double category. As suggested by the notation, its vertical category and vertical cell category are $\pm_v$ and $\pm_V$. Its horizontal category $\pm_h$ is generated by the graph $O \to P$. It is thus isomorphic to the ordinal $\mathbb{2}$: there are exactly three horizontal morphisms: OO, PP, and OP. To complete the definition, it remains to define composition and identities in $\pm_H$. One way is to depict basic cells as triangles



(7)

respectively denoting $OO \to OP$, $OP \to PP$, $PP \to OP$, and $OP \to OO$. General cells are obtained by stacking up such basic triangles. Depicting cells as stacks of triangles yields the following inductive definition of composition of cells α and β as in (6):

- If there is an 'outwards' bottom triangle, i.e., the bottom of α and β look like either of

  with $X \in \{O, P\}$ and $X^\perp$ denoting the other player, then the composite is obtained by composing the rest of α and β, and appending the obvious triangle ($(OX, OX^\perp)$, resp. $(XP, X^\perp P)$).
- Otherwise, there is a pair of interacting bottom triangles, as below left, in which case the composite is simply the composite of the rest of α and β – which is precisely where game semantical *hiding* is encoded in $\pm$.

▶ Remark 44. Ambiguous configurations as above right, where we would not know which triangle to put last in the composite, cannot occur. Indeed, existence of the left-hand triangle forces $M = P$, while existence of the right-hand one forces $M = O$.

Horizontal identities are the so-called *copycat* schedules. The copycat on the vertical morphism $O \to P$ is obtained by composing $OO \to OP$ and $OP \to PP$, and dually for $P \to O$. The copycat schedule of a general morphism is the obvious composite of these basic copycats.

We obtain as promised:

▶ **Proposition 45** (Melliès [14]). *The template $\pm$ forms a double category.*

4.3 Template games as a double slice

There is an alternative point of view on double categories which will be crucial to us: they may be axiomatised based on a span of functors $\mathcal{C}_v \xleftarrow{s} \mathcal{C}_V \xrightarrow{t} \mathcal{C}_v$. For this, let us consider the following structure, which is almost a (large) double category.

▶ **Definition 46.** *Let* Span(Cat) *have as objects all small categories, as vertical morphisms all functors, as horizontal morphisms $A \to B$ all spans $A \leftarrow C \to B$ of functors, and as cells below left all commuting diagrams as below right in* Cat.

Vertical composition is given by (componentwise) composition of functors, while horizontal composition is given by pullbacks and their universal property.

The structure formed by Span(Cat) is a *weak double category* [6], a weak form of double category where horizontal composition is only associative and unital up to coherent isomorphism, in a suitable sense.

▶ Remark 47. The horizontal arrows and *special* cells of a weak double category $\mathcal{C}$ form a bicategory $\mathcal{H}(\mathcal{C})$, where special means that the left and right borders are identities.

▶ Remark 48. The reason Span(Cat) is weak is that one cannot hope to make a strictly associative choice of pullbacks.

Just like one usually does in bicategories, we may define monads internally to weak double categories.

▶ **Definition 49.** *A* monad *in a weak double category $\mathcal{C}$ is a horizontal morphism $M\colon X \to X$, equipped with special cells*

satisfying the obvious generalisation of the usual monad laws.

▶ **Proposition 50.** *A double category is the same as a monad in* Span(Cat)*.*

Proof. Composing an endo-span $\mathcal{C}_v \xleftarrow{s} \mathcal{C}_V \xrightarrow{t} \mathcal{C}_v$ with itself in Span(Cat) amounts to constructing the category of pairs of compatible horizontal morphisms and cells (6), so requiring a monad multiplication is requiring horizontal composition. Similarly, requiring a monad unit amounts to requiring horizontal identities. ◀

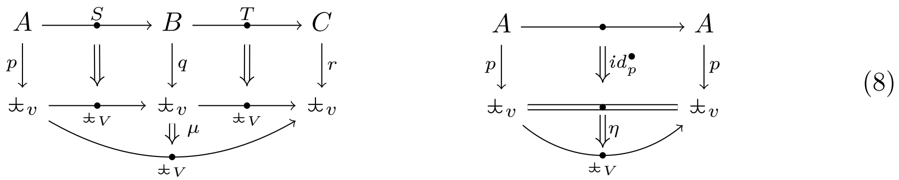
Explicitly, if $\mathcal{C}$ is a double category, then it can be seen as a monad $\mathcal{C}_V : \mathcal{C}_v \to \mathcal{C}_v$ in Span(Cat) with μ and η given by horizontal composition and identities.

It should now be clear that a template game is a vertical morphism to $\pm_v$ in Span(Cat), while a template strategy $A \to B$ is merely a cell

$$\begin{array}{ccc} A & \xrightarrow{S} & B \\ {\scriptstyle p}\downarrow & \Downarrow & \downarrow{\scriptstyle q} \\ \pm_v & \xrightarrow[\pm_V]{} & \pm_v . \end{array}$$

This allows us to define composition of template strategies, using the monad structure of $\pm$ given by Propositions 50 and 45.

▶ **Proposition 51.** *The composite of $S\colon A \to B$ and $T\colon B \to C$ in the sense of [14] is the pasting below left, while the identity on any $p\colon A \to \pm_v$ is the one below right.*



(8)

From this, it easily follows that strategies in fact form a weak double category, and clearly this construction works for any monad in any weak double category. Namely, Melliès's construction may be obtained by applying the following theorem.

▶ **Theorem 52.** *Given any monad $M_V\colon M_v \to M_v$ in a weak double category $\mathcal{C}$, there is a* slice *weak double category $\mathcal{C}/M$ whose*

- *vertical category is* $(\mathcal{C}/M)_v = \mathcal{C}_v/M_v$*;*
- *vertical cell category is* $(\mathcal{C}/M)_V = \mathcal{C}_V/M_V$*;*
- *horizontal composition is given by pasting with* μ*, as on the left in* (8)*;*
- *horizontal identity on* $p\colon A \to M_v$ *is by pasting with* η *as on the right in* (8)*.*
- *and all operations on cells are given by their counterparts in* $\mathcal{C}$*.*

▶ **Proposition 53.** *The weak double category* $\underline{\mathbf{Games}}(\pm)$ *of template games [14] is equal to* $\mathsf{Span}(\mathsf{Cat})/\pm$*.*

A weak double category with trivial category $\mathcal{C}_v$ is nothing but a monoidal category. In that case, the theorem reduces to the following well-known result used, e.g., in Weber [17]:

▶ **Corollary 54.** *The slice of a monoidal category over a monoid is again monoidal.*

5 Refined template games, simple games, and Day convolution

5.1 Refined template games

We now want to recover simple games by replacing $\pm$ with $\mathbb{T}$, and restricting the slice construction in $\mathsf{Span}(\mathsf{Cat})$ to discrete fibrations (for vertical morphisms and cells). For this, we appeal to factorisation systems.

▶ **Definition 55.** *For all morphisms* f *and* g *in a category* $\mathcal{C}$*, let* $f \perp g$ *iff for all commuting squares as in the solid part of*

$$\begin{array}{ccc} A & \longrightarrow & C \\ {\scriptstyle f}\downarrow & \overset{k}{\nearrow} & \downarrow{\scriptstyle g} \\ B & \longrightarrow & D \end{array}$$

there exists a unique k *as shown making both triangles commute. For any class* $\mathcal{M}$ *of morphisms, let* $\mathcal{M}^{\perp} = \{g \mid \forall f \in \mathcal{M}, f \perp g\}$*. We define* ${}^{\perp}\mathcal{M}$ *symmetrically.*

A (strong) factorisation system *[1] on a category* $\mathcal{C}$ *consists in classes* $\mathcal{L}$ *and* $\mathcal{R}$ *of arrows such that* $\mathcal{L}^{\perp} = \mathcal{R}$*,* $\mathcal{L} = {}^{\perp}\mathcal{R}$*, and every arrow factors as* $r \circ l$ *with* $l \in \mathcal{L}$ *and* $r \in \mathcal{R}$*.*

▶ **Example 56.** Discrete fibrations form the right class of the *comprehensive* factorisation system $(\mathsf{Fin}, \mathsf{DFib})$ on Cat, whose left class is that of *final* functors [16].

Our refined construction is based on the following generalisation of factorisation systems.

▶ **Definition 57.** *A* double factorisation system *on a weak double category* $\mathcal{C}$ *consists of factorisation systems* $(\mathcal{L}_v, \mathcal{R}_v)$ *and* $(\mathcal{L}_V, \mathcal{R}_V)$ *on* $\mathcal{C}_v$ *and* $\mathcal{C}_V$*, respectively, such that the source and target functors* $\mathcal{C}_V \to \mathcal{C}_v$ *both map* $\mathcal{L}_V$ *to* $\mathcal{L}_v$ *and* $\mathcal{R}_V$ *to* $\mathcal{R}_v$*, and all cells* α *as below left with* $l, l' \in \mathcal{L}_v$ *and* $r, r' \in \mathcal{R}_v$ *factor as below right, with* $\lambda \in \mathcal{L}_V$ *and* $\rho \in \mathcal{R}_V$*.*

$$\begin{array}{ccc} A & \xrightarrow{S} & A' \\ {\scriptstyle l}\downarrow & & \downarrow{\scriptstyle l'} \\ B & \Downarrow{\scriptstyle \alpha} & B' \\ {\scriptstyle r}\downarrow & & \downarrow{\scriptstyle r'} \\ C & \xrightarrow[U]{} & C' \end{array} \quad = \quad \begin{array}{ccc} A & \xrightarrow{S} & A' \\ {\scriptstyle l}\downarrow & \Downarrow{\scriptstyle \lambda} & \downarrow{\scriptstyle l'} \\ B & \xrightarrow{T} & B' \\ {\scriptstyle r}\downarrow & \Downarrow{\scriptstyle \rho} & \downarrow{\scriptstyle r'} \\ C & \xrightarrow[U]{} & C' \end{array}$$

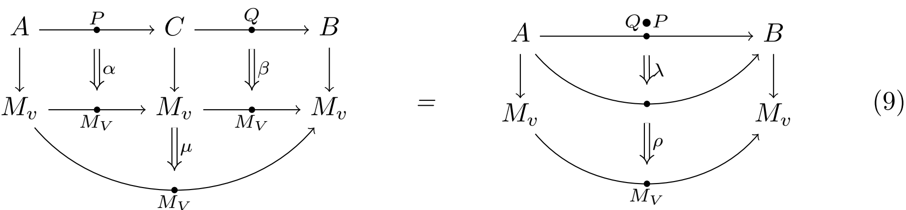
▶ **Lemma 58.** *Discrete fibrations and componentwise discrete fibrations are the right classes of a double factorisation system* $((\mathsf{Fin}, \mathsf{DFib}), (\mathsf{Fin}_V, \mathsf{DFib}_V))$ *on* $\mathsf{Span}(\mathsf{Cat})$*. Furthermore,* DFib_V *is stable under horizontal composition and identities.*

Proof. As is well-known, discrete fibrations may be defined by unique lifting w.r.t. the injection $\mathbb{1} \hookrightarrow \mathbb{2}$ mapping 0 to 1. Componentwise discrete fibrations may be defined similarly, in $\mathsf{Span}(\mathsf{Cat})_V$. The result then follows from componentwise discrete fibrations being stable under horizontal composition, which holds because they are stable under pullback in the arrow category. ◀

In order to state the promised generalisation of Theorem 52 in full generality, we need the following, somewhat awkward notion of stability of left residuals. Indeed, our two applications work for rather different reasons, as emphasised by Corollaries 62 and 63 below.

▶ **Definition 59.** *A monad* $M_V \colon M_v \to M_v$ *in a weak double category* $\mathcal{C}$ *has* stable left residuals *w.r.t. a double factorisation system* $((\mathcal{L}_v, \mathcal{R}_v), (\mathcal{L}_V, \mathcal{R}_V))$*, iff*

(a) *for all* $\alpha, \beta \in \mathcal{R}_V$ *such that the composite below left factors as on the right,*

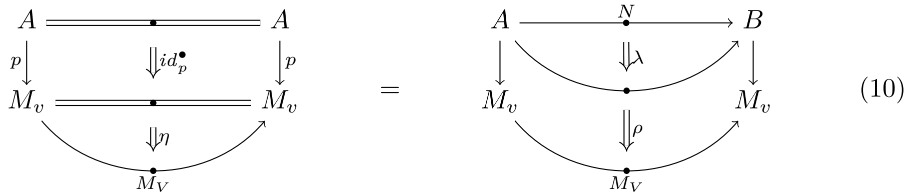


(9)

for any $S \colon A' \to A$ *and* $T \colon B \to B'$ *in* $\mathcal{C}_h$*, the composite* $id_T \bullet \lambda \bullet id_S$ *below is in* $\mathcal{L}_V$*;*

$$A' \xrightarrow{S} A \;\Downarrow\lambda\; B \xrightarrow{T} B',$$

(b) *for all* $p \in \mathcal{R}_v$ *such that the composite below left factors as on the right, for any* $S \colon A' \to A$ *and* $T \colon B \to B'$ *in* $\mathcal{C}_h$*,* $id_T \bullet \lambda \bullet id_S$ *is in* $\mathcal{L}_V$*.*



(10)

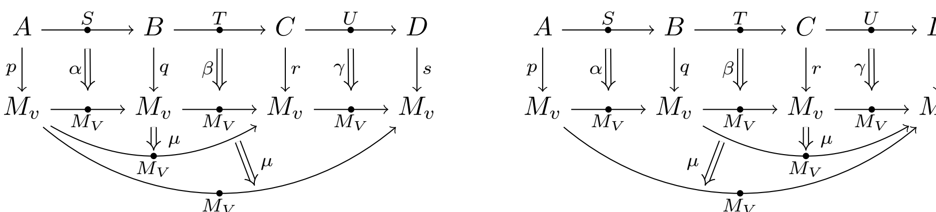
▶ Remark 60. In both cases, existence of a special λ follows by Definition 57.

▶ **Theorem 61.** *Consider any monad* $M_V \colon M_v \to M_v$ *in a weak double category* $\mathcal{C}$ *with stable left residuals w.r.t. a double factorisation system* $((\mathcal{L}_v, \mathcal{R}_v), (\mathcal{L}_V, \mathcal{R}_V))$*. Then there is a* slice *weak double category* $\mathcal{C}/_{\mathcal{R}_V} M$ *whose*

- *vertical category* $(\mathcal{C}/_{\mathcal{R}_V} M)_v$ *is* $\mathcal{C}_v/_{\mathcal{R}_v} M_v$*, the full subcategory of* $\mathcal{C}_v/M_v$ *on maps in* $\mathcal{R}_v$*;*
- *vertical category of cells is* $(\mathcal{C}/_{\mathcal{R}_V} M)_V = \mathcal{C}_V/_{\mathcal{R}_V} M_V$*;*
- *horizontal composition is given by* ρ *in* (9)*;*
- *horizontal identity on any* $p \colon A \to M_v$ *is given by* ρ *in* (10)*;*
- *and all operations on cells are given by their counterparts in* $\mathcal{C}$*.*

Proof. By coherence for weak double categories [8, Theorem 7.5], and assuming a higher universe in which $\mathcal{C}$ is small, we may assume that $\mathcal{C}$ is in fact a (strict) double category.

Composition of cells in $\mathcal{C}/_{\mathcal{R}_V}M$ is just as in $\mathcal{C}$, so the only non-trivial point to check is weak associativity and unitality of horizontal composition (of morphisms). For weak associativity, we observe that both cells



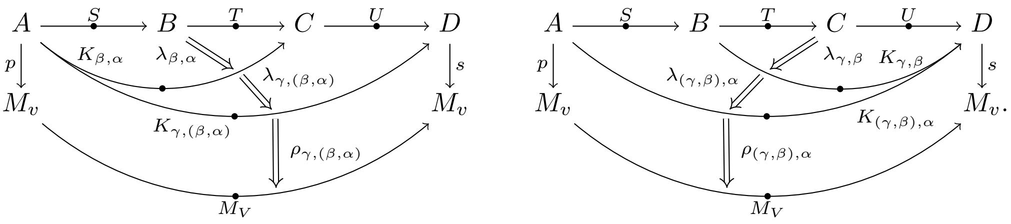
are equal. Now, denoting composition in $(\mathcal{C}/_{\mathcal{R}_V}M)_h$ by $\tilde{\bullet}$, $\gamma\tilde{\bullet}(\beta\tilde{\bullet}\alpha)$ and $(\gamma\tilde{\bullet}\beta)\tilde{\bullet}\alpha$ are obtained by factoring them as follows. For the former, we factor

$$T \bullet S \xrightarrow{\beta\bullet\alpha} M_V \bullet M_V \xrightarrow{\mu} M_V \qquad \text{as} \qquad T \bullet S \xrightarrow{\lambda_{\beta,\alpha}} K_{\beta,\alpha} \xrightarrow{\rho_{\beta,\alpha}} M_V,$$

in which $\lambda_{\beta,\alpha}$ special by definition. We then factor

$$U \bullet K_{\beta,\alpha} \xrightarrow{\gamma\bullet\rho_{\beta,\alpha}} M_V \bullet M_V \xrightarrow{\mu} M_V \qquad \text{as} \qquad U \bullet K_{\beta,\alpha} \xrightarrow{\lambda_{\gamma,(\beta,\alpha)}} K_{\gamma,(\beta,\alpha)} \xrightarrow{\rho_{\gamma,(\beta,\alpha)}} M_V.$$

The other composite may be computed symmetrically, so that we obtain factorisations:



By stability of left residuals, both are in fact factorisations for $(\mathcal{L}_V, \mathcal{R}_V)$, so that by lifting, we obtain a special cell $a_{\alpha,\beta,\gamma} \colon K_{\gamma,(\beta,\alpha)} \xrightarrow{\simeq} K_{(\gamma,\beta),\alpha}$ such that $\rho_{(\gamma,\beta),\alpha} \circ a_{\alpha,\beta,\gamma} = \rho_{\gamma,(\beta,\alpha)}$, which is our candidate associator for $\mathcal{C}/_{\mathcal{R}_V}M$. It satisfies the MacLane pentagon by uniqueness of lifting.

Weak unitality follows similarly. ◀

▶ **Corollary 62.** *Consider any monad $M_V \colon M_v \twoheadrightarrow M_v$ in a weak double category $\mathcal{C}$ with double factorisation system $((\mathcal{L}_v, \mathcal{R}_v), (\mathcal{L}_V, \mathcal{R}_V))$. If $\eta, \mu \in \mathcal{R}_V$ and $\mathcal{R}_V$ is stable under horizontal composition and identities, then $\mathcal{C}/_{\mathcal{R}_V}M$ exists and is a sub weak double category of $\mathcal{C}/M$.*

Proof. Both pastings on the left of (9) and (10) are already in $\mathcal{R}_V$. ◀

▶ **Corollary 63.** *Consider any monad $M_V \colon M_v \twoheadrightarrow M_v$ in a weak double category $\mathcal{C}$ with double factorisation system $((\mathcal{L}_v, \mathcal{R}_v), (\mathcal{L}_V, \mathcal{R}_V))$. If $\mathcal{L}_V$ is stable under horizontal composition, then $\mathcal{C}/_{\mathcal{R}_V}M$ is a weak double category.*

Proof. Stability under horizontal composition entails stability under whiskering. ◀

5.2 Simple games

Let us at last return to simple games. We have:

▶ **Lemma 64.** *$\mathbb{T}$ is a monad whose multiplication and unit are in DFib_V.*

Proof. In $\mathbb{T}$, multiplication is composition of schedules (through parallel composition and hiding) and the unit is given by copycat schedules. The crucial point to prove that multiplication is in DFib_V is that, for any pair (p, q) of schedules in $\mathbb{T}_V \bullet \mathbb{T}_V$, the last move in $s(q)$ $(= t(p))$ cannot be last in both p and q for polarity reasons. For the unit, the crucial point is that copycat schedules are closed under restrictions. ◀

By Corollary 62, we have:

▶ **Corollary 65.** $\mathsf{Span}(\mathsf{Cat})/_{\mathsf{DFib}_V}\mathbb{T}$ *is a sub weak double category of* $\mathsf{Span}(\mathsf{Cat})$*.*

Finally, let us relate to simple strategies.

▶ **Definition 66** ([12, Definition 9])**.** *Simple games, strategies, and natural transformations form a bicategory $\mathcal{S}$.*

▶ **Theorem 67.** *The full, reflective embeddings $F_{A,B} \colon \mathcal{S}(A, B) \rightsquigarrow \mathsf{DFib}((A \to B)^P)$ of Corollary 43 determine a locally reflective and fully-faithful weak 2-functor*

$$\mathcal{S} \to \mathcal{H}(\mathsf{Span}(\mathsf{Cat})/_{\mathsf{DFib}_V}\mathbb{T})$$

(where $\mathcal{H}$ is the bicategory of special cells, as in Remark 47), whose essential image consists of definite refined template games and strategies.

Proof. The essential image part of the result is clear. By [13, §2.2], we need to organise the full, reflective embeddings $F_{A,B}$ into a weak 2-functor, which here means that F commutes with composition of strategies up to coherent isomorphism. ◀

▶ **Corollary 68.** *The bicategory $\mathcal{S}$ of simple games and strategies is biequivalent to the locally full sub-bicategory of $\mathcal{H}(\mathsf{Span}(\mathsf{Cat})/_{\mathsf{DFib}_V}\mathbb{T})$ spanning definite refined template games and strategies.*

5.3 Day convolution

We finally reach the application mentioned in the introduction: Day convolution. The purpose of this operation is to show that $\widehat{\mathbb{C}}$ is monoidal when $\mathbb{C}$ is. Let us now recover this structure from Theorem 61, in the particular case where $\mathbb{C}$ is strictly monoidal. The starting point is the following weak double category:

▶ **Definition 69.** *Let $\mathcal{W}$ be the sub weak double category of* $\mathsf{Span}(\mathsf{Cat})$ *obtained by restricting objects to the terminal object $\mathbb{1}$ and vertical morphisms to the identity thereon.*

Thus, $\mathcal{W}_V$ consists of categories and functors, and horizontal composition is given by cartesian product and its universal property.

▶ **Lemma 70.** *A monad in $\mathcal{W}$ is a strict monoidal category $\mathbb{C}$.*

Proof. The monad multiplication $\mu\colon \mathbb{C}\times\mathbb{C}\to\mathbb{C}$ gives the tensor product while the unit $\eta\colon \mathbb{1}\to\mathbb{C}$ gives the tensor unit, and coherence equations for the monad those of the monoidal category. ◀

Clearly, final functors and discrete fibrations form a double factorisation system $(\mathcal{L}_V, \mathcal{R}_V) = (\mathsf{Fin}, \mathsf{DFib})$ on $\mathcal{W}$. Final functors being closed under binary products, $\mathcal{L}_V$ is closed under horizontal composition and identities, so Corollary 63 applies and we obtain a weak double category $\mathcal{W}/_{\mathsf{DFib}}\mathbb{C}$. This weak double category is vertically trivial, hence underlies a monoidal category, say $\mathbb{C}'$.

Let us now show that $\mathbb{C}'$ is equivalent to $\widehat{\mathbb{C}}$ equipped with the convolution tensor product. We first recall that the latter is given as follows:

▶ **Definition 71.** *For any small monoidal category $\mathbb{C}$ and $X, Y \in \widehat{\mathbb{C}}$, let*

$$(X\otimes Y)(c) = \int^{(c_1,c_2)\in\mathbb{C}^2} X(c_1)\times Y(c_2)\times \mathbb{C}(c, c_1\otimes c_2).$$

▶ **Lemma 72.** *Let $f\colon A\to B$ be a functor. The discrete fibration ρ_f associated to f by the comprehensive factorisation system is determined up to isomorphism by*

$$\partial^\star(\rho_f)(b)\cong\int^{a\in A} B(b, f(a)),$$

where $\partial^\star\colon \mathsf{DFib}(B)\to\widehat{B}$ is the standard equivalence between discrete fibrations and presheaves.

Proof. This is actually obvious by construction. In [16], the dual case is actually treated, initial functors and discrete opfibrations. But up to this discrepancy, $\partial^\star(\rho_f)$ is precisely k in the proof of [16, Theorem 3], which would in our case be defined as the left Kan extension of $A^{op}\xrightarrow{!}1\xrightarrow{1}\mathsf{Set}$ along f^{op}. By the well-known characterisation of left Kan extensions by coends, we readily obtain the desired formula. ◀

▶ **Theorem 73.** *For any strictly monoidal category $\mathbb{C}$, the monoidal category $\mathbb{C}'$ is equivalent to $\widehat{\mathbb{C}}$ equipped with the convolution tensor product.*

Proof. By construction, given two presheaves $X, Y\in\widehat{\mathbb{C}}$ and transporting them to their corresponding discrete fibrations, say $S\colon \mathrm{el}(X)\to\mathbb{C}$ and $T\colon \mathrm{el}(Y)\to\mathbb{C}$, their tensor product $S\tilde{\bullet}T$ in $\mathbb{C}'$ is the right factor of the composite

$$\mathrm{el}(X)\times\mathrm{el}(Y)\xrightarrow{S\times T}\mathbb{C}\times\mathbb{C}\xrightarrow{\otimes}\mathbb{C}.$$

By Lemma 72, the result has its corresponding presheaf defined up to isomorphism by

$$\begin{array}{rcl}\partial^\star(S\tilde{\bullet}T)(c) & \cong & \int^{(a,b)\in\mathrm{el}(X)\times\mathrm{el}(Y)}\mathbb{C}(c,\otimes((S\times T)(a,b)))\\ & = & \int^{(a,b)\in\mathrm{el}(X)\times\mathrm{el}(Y)}\mathbb{C}(c,S(a)\otimes T(b))\\ & \cong & \int^{((c_1,x),(c_2,y))\in\mathrm{el}(X)\times\mathrm{el}(Y)}\mathbb{C}(c,c_1\otimes c_2)\\ & \cong & \int^{c_1,c_2} X(c_1)\times Y(c_2)\times\mathbb{C}(c,c_1\otimes c_2),\end{array}$$

as desired. ◀

6 Conclusion and perspectives

We have designed an abstract slice construction over monads in weak double categories, which has as instances (1) a weak double category of simple games and non-deterministic strategies, whose underlying bicategory of definite games and strategies is biequivalent to the standard one, and (2) the monoidal category of presheaves over any small strict monoidal category.

For future work, we first could try to accomodate not only the weak double category structure of template games, but also symmetric monoidal closedness. Furthermore, Melliès is also currently working on a template game model of full linear logic. This will of course be a useful feature to incorporate to our framework. Another direction for future work is to generalise our construction to encompass Day convolution for non-strict monoidal categories. What is needed here is a common generalisation of [7, Theorem 14.5] and Theorem 61. Finally, it is slightly unsatisfactory to only get standard simple games up to a locally fully-faithful embedding. In order to obtain a biequivalence, preliminary investigation suggests that instead of restricting the slice construction w.r.t. some factorisation system, it would be more expressive to construct the factorisation system directly in the relevant slice, for which fibrant objects would be the desired strategies. Indeed, e.g., polarities, which cannot be used before slicing, become available in the slice. This technique also seems to apply for refining the correspondence to, e.g., deterministic strategies, and possibly even innocence.

References

1. A. K. Bousfield. Constructions of Factorization Systems in Categories. *Journal of Pure and Applied Algebra*, 9(2-3):287–329, 1977.
2. B. Day. On closed categories of functors. In *Reports of the Midwest Category Seminar IV*, volume 137 of *Lecture Notes in Mathematics*, pages 1–38. Springer, 1970.
3. C. Eberhart. *Catégories et diagrammes de cordes pour les jeux concurrents.* PhD thesis, Université Savoie Mont Blanc, 2018.
4. C. Eberhart and T. Hirschowitz. What's in a game?: A theory of game models. In *Proc. 33rd Symposium on Logic in Computer Science*, pages 374–383. ACM, 2018. doi:10.1145/3209108.3209114.
5. C. Ehresmann. Catégories structurées. *Annales scientifiques de l'Ecole Normale Supérieure*, 80(4):349–426, 1963.
6. R. Garner. *Polycategories.* PhD thesis, University of Cambridge, 2006.
7. R. Garner and M. Shulman. Enriched categories as a free cocompletion. *Advances in Mathematics*, 289:1–94, 2016.
8. M. Grandis and R. Paré. Limits in double categories. *Cahiers de Topologie et Géométrie Différentielle Catégoriques*, 40(3):162–220, 1999.
9. R. Harmer, J. M. E. Hyland, and P.-A. Melliès. Categorical Combinatorics for Innocent Strategies. In *Proc. 22nd Symposium on Logic in Computer Science*, pages 379–388. IEEE, 2007.
10. M. Hirschowitz, A. Hirschowitz, and T. Hirschowitz. A theory for game theories. In *Proc. 27th Foundations of Software Technology and Theoretical Computer Science*, volume 4855 of *Lecture Notes in Computer Science*, pages 192–203. Springer, 2007.
11. J. M. E. Hyland. Game Semantics. In Andrew M. Pitts and Peter Dybjer, editors, *Semantics and Logics of Computation*, pages 131–184. Cambridge University Press, 1997.
12. C. Jacq and P.-A. Melliès. Categorical Combinatorics for Non Deterministic Strategies on Simple Games. In *Proc. 21st Foundations of Software Science and Computational Structures*, volume 10803 of *Lecture Notes in Computer Science*, pages 39–70. Springer, 2018. doi:10.1007/978-3-319-89366-2_3.

13 T. Leinster. Basic Bicategories. *arXiv Mathematics e-prints*, page math/9810017, October 1998. `arXiv:math/9810017`.

14 P.-A. Melliès. Categorical combinatorics of scheduling and synchronization in game semantics. *Proc. ACM Program. Lang.*, 3(POPL):23:1–23:30, January 2019. `doi:10.1145/3290336`.

15 S. Rideau and G. Winskel. Concurrent Strategies. In *Proc. 26th Symposium on Logic in Computer Science*, pages 409–418. IEEE, 2011.

16 R. Street and R. F. C. Walters. The Comprehensive Factorization of a Functor. *Bulletin of the American Mathematical Society*, 79(5), 1973.

17 M. Weber. Generic morphisms, parametric representations and weakly cartesian monads. *Theory and Applications of Categories*, 13:191–234, 2004.

Differentials and Distances in Probabilistic Coherence Spaces

Thomas Ehrhard
CNRS, IRIF, Université de Paris, France
https://www.irif.fr/~ehrhard/
ehrhard@irif.fr

Abstract

In probabilistic coherence spaces, a denotational model of probabilistic functional languages, morphisms are analytic and therefore smooth. We explore two related applications of the corresponding derivatives. First we show how derivatives allow to compute the expectation of execution time in the weak head reduction of probabilistic PCF (pPCF). Next we apply a general notion of "local" differential of morphisms to the proof of a Lipschitz property of these morphisms allowing in turn to relate the observational distance on pPCF terms to a distance the model is naturally equipped with. This suggests that extending probabilistic programming languages with derivatives, in the spirit of the differential lambda-calculus, could be quite meaningful.

2012 ACM Subject Classification Theory of computation → Lambda calculus; Theory of computation → Probabilistic computation; Theory of computation → Abstract machines; Theory of computation → Linear logic

Keywords and phrases Denotational semantics, probabilistic coherence spaces, differentials of programs

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.17

Acknowledgements We thank Raphaëlle Crubillé, Paul-André Melliès, Michele Pagani and Christine Tasson for many enlightening discussions on this work. We also thank the referees for their precious comments and suggestions.

Introduction

Currently available denotational models of probabilistic functional programming (with full recursion, and thus partial computations) can be divided in three classes.

- *Game* based models, first proposed in [6] and further developed by various authors (see [2] for an example of this approach). From their deterministic ancestors they typically inherit good definability features.
- Models based on Scott continuous functions on domains endowed with additional probability related structures. Among these models we can mention *Kegelspitzen* [13] (domains equipped with an algebraic convex structure) and *ω-quasi Borel spaces* [15] (domains equipped with a generalized notion of measurability), this latter semantics, as far as we understand the situation, requiring the use of an adapted probabilistic powerdomain construction.
- Models based on (a generalization of) Berry stable functions. The first category of this kind was that of *probabilistic coherence spaces* (PCSs) and power series with non-negative coefficients (the Kleisli category of the model of Linear Logic developed in [5]) for which we could prove adequacy and full abstraction with respect to a probabilistic version of PCF [10]. We extended this idea to "continuous data types" (such as $\mathbb{R}$) by substituting PCSs with *positive cones* and power series with functions featuring

© Thomas Ehrhard;
licensed under Creative Commons License CC-BY
4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019).
Editor: Herman Geuvers; Article No. 17; pp. 17:1–17:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

an hereditary monotonicity property that we called *stability*[1] and [3] showed that this extension is actually conservative (stable functions on PCSs, which are special positive cones, are exactly power series).

The main feature of this latter semantics is the extreme regularity of its morphisms. Being power series, they must be smooth. Nevertheless, the category **Pcoh** is not a model of differential linear logic in the sense of [9]. This is due to the fact that general addition of morphisms is not possible (only sub-convex linear combinations are available) thus preventing, *e.g.*, the Leibniz rule to hold in the way it is presented in differential LL. Also a morphism $X \to Y$ in the Kleisli category $\mathbf{Pcoh}_!$ can be considered as a function from the *closed unit ball* of the cone P associated with X to the closed unit ball of the cone Q associated with Y. From a differential point of view such a morphism is well behaved only in the interior of the unit ball. On the border derivatives can typically take infinite values.

Contents

We already used the analyticity of the morphisms of $\mathbf{Pcoh}_!$ to prove full abstraction results [10]. We provide here two more corollaries of this properties, involving now also derivatives. For both results, we consider a paradigmatic probabilistic purely functional programming language[2] which is a probabilistic extension of Scott and Plotkin's PCF. This language pPCF features a single data type ι of integers, a simple probabilistic choice operator $\mathsf{coin}(r) : \iota$ which flips a coin with probability r to get $\underline{0}$ and $1 - r$ to get $\underline{1}$. To make probabilistic programming possible, this language has a $\mathsf{let}(x, M, N)$ construct restricted to M of type ι which allows to sample an integer according to the sub-probability distribution represented by M. The operational semantics is presented by a deterministic "stack machine" which is an environment-free version of Krivine machine parameterized by a choice sequence $\in \mathcal{C}_0 = \{0,1\}^{<\omega}$, presented as a partial *evaluation function.* We adopt a standard discrete probability approach, considering $\mathcal{C}_0$ as our basic sample space and the evaluation function as defining a (total) probability density function on $\mathcal{C}_0$. We also introduce an extension $\mathsf{pPCF}_{\mathsf{lab}}$ of pPCF where terms can be labeled by elements of a set $\mathcal{L}$ of labels, making it possible to count the use of labeled subterms during a reduction. Evaluation for this extended calculus gives rise to a random variable (r.v.) on $\mathcal{C}_0$ ranging in the set $\mathcal{M}_{\mathrm{fin}}(\mathcal{L})$ of finite multisets of elements of $\mathcal{L}$. The number of uses of terms labeled by a given $l \in \mathcal{L}$ (which is a measure of the computation time) is then an $\mathbb{N}$-valued r.v. the expectation of which we want to evaluate. We prove that, for a given labeled closed term M of type ι, this expectation can be computed by taking a derivative of the interpretation of this term in the model $\mathbf{Pcoh}_!$ and provide a concrete example of computation of such expectations. This result can be considered as a probabilistic version of [7, 8]. The fact that derivatives can become infinite on the border of the unit ball corresponds then to the fact that this expectation of "computation time" can be infinite.

In the second application, we consider the contextual distance on pPCF terms generalizing Morris equivalence as studied in [4] for instance. The probabilistic features of the language make this distance too discriminating, putting *e.g.* terms $\mathsf{coin}(0)$ and $\mathsf{coin}(\varepsilon)$ at distance 1 for all $\varepsilon > 0$ (*probability amplification*). Any cone (and hence any PCS) is equipped with a norm and hence a canonically defined metric. Using a *locally defined* notion of differential

[1] Because, when reformulated in the domain-theoretic framework of Girard's coherence spaces, this condition exactly characterizes Berry's stable functions.

[2] One distinctive feature of our approach is to not consider probabilities as an effect.

of morphisms in $\mathbf{Pcoh}_!$, we prove that these morphisms enjoy a Lipschitz property on all balls of radius $p < 1$, with a Lipschitz constant $1/(1-p)$ (thus tending towards ∞ when p tends towards 1). Modifying the definition of the operational distance by not considering all possible contexts, but only those which "perturb" the tested terms by allowing them to diverge with probability $1-p$, we upper bound this p-tamed distance by the distance of the model with a ratio $p/(1-p)$. Being in some sense defined wrt. *linear* semantic contexts, the denotational distance does not suffer from the probability amplification phenomenon. This suggests that p-tamed distances might be more suitable than ordinary contextual distances to reason on probabilistic programs.

Notations

We use $\mathbb{R}_{\geq 0}$ for the set of real numbers x such that $x \geq 0$, and we set $\overline{\mathbb{R}_{\geq 0}} = \mathbb{R}_{\geq 0} \cup \{+\infty\}$. Given two sets S and I we use S^I for the set of functions $I \to S$, often considered as I-indexed families $\vec{s}$ of elements of S (the purpose of the arrow is to stress the fact that this object is such a family), the indexing set I being usually easily derivable from the context. The elements of such a family $\vec{s}$ are denoted s_i or $s(i)$ depending on the context. Given $i \in I$ we use $\overline{i}$ for the function $I \to \mathbb{R}_{\geq 0}$ such that $\overline{i}(i) = 1$ and $\overline{i}(j) = 0$ if $j \neq i$. We use $\mathcal{M}_{\mathrm{fin}}(I)$ for the set of finite multisets of elements of I. Such a multiset is a function $\mu : I \to \mathbb{N}$ such that $\mathsf{supp}(\mu) = \{i \in I \mid \mu(i) \neq 0\}$ is finite. We use additive notations for operations on multisets (0 for the empty multiset, $\mu + \nu$ for their pointwise sum). We use $[i_1, \dots, i_k]$ for the multiset μ such that $\mu(i) = \#\{j \in \mathbb{N} \mid i_j = i\}$. If $\mu, \nu \in \mathcal{M}_{\mathrm{fin}}(I)$ with $\mu \leq \nu$ (pointwise order), we set $\binom{\nu}{\mu} = \prod_{i \in I} \binom{\nu(i)}{\mu(i)}$ where $\binom{n}{m} = \frac{n!}{m!(n-m)!}$ is the usual binomial coefficient. We use $I^{<\omega}$ for the set of finite sequences $\langle i_1, \dots, i_k \rangle$ of elements of I and $\alpha\,\beta$ for the concatenation of such sequences. We use $\langle\rangle$ for the empty sequence.

1 Probabilistic coherence spaces (PCS)

For the general theory of PCSs we refer to [5, 10]. We recall briefly the basic definitions and provide a characterization of these objects. PCSs are particular cones (a notion borrowed from [14]) as we used them in [10], so we start with a few words about these more general structures to which we plan to extend the constructions of this paper.

1.1 A few words about cones

A (positive) *pre-cone* is a cancellative[3] commutative $\mathbb{R}_{\geq 0}$-semi-module P equipped with a norm $\|_\|_P$, that is a map $P \to \mathbb{R}_{\geq 0}$, such that $\|r\,x\|_P = r\,\|x\|_P$ for $r \in \mathbb{R}_{\geq 0}$, $\|x+y\|_P \leq \|x\|_P + \|y\|_P$ and $\|x\|_P = 0 \Rightarrow x = 0$. It is moreover assumed that $\|x\|_P \leq \|x+y\|_P$, this condition expressing that the elements of P are positive. Given $x, y \in P$, one says that x is less than y (notation $x \leq y$) if there exists $z \in P$ such that $x + z = y$. By the cancellation property, if such a z exists, it is unique and we denote it as $y - x$. This subtraction obeys usual algebraic laws (when it is defined). Notice that if $x, y \in P$ satisfy $x + y = 0$ then since $\|x\|_P \leq \|x+y\|_P$, we have $x = 0$ (and of course also $y = 0$). Therefore, if $x \leq y$ and $y \leq x$ then $x = y$ and so $\leq$ is an order relation.

A (positive) *cone* is a positive pre-cone P whose unit ball $\mathcal{B}P = \{x \in P \mid \|x\|_P \leq 1\}$ is ω-order-complete in the sense that any increasing sequence of elements of $\mathcal{B}P$ has a least upper bound in $\mathcal{B}P$. In [10] we show how a notion of *stable* function on cones can be defined, which gives rise to a cartesian closed category.

[3] Meaning that $x + y = x' + y \Rightarrow x = x'$.

The following construction will be crucial in Section 3.2. Given a cone P and $x \in \mathcal{B}P$, we define the *local cone at* x as the set $P_x = \{u \in P \mid \exists \varepsilon > 0\ x + \varepsilon u \in \mathcal{B}P\}$. Equipped with the algebraic operations inherited from P, this set is clearly a $\mathbb{R}_{\geq 0}$-semi-ring. We equip it with the following norm: $\|u\|_{P_x} = \inf\{\varepsilon^{-1} \mid \varepsilon > 0 \text{ and } x + \varepsilon u \in \mathcal{B}P\}$ and then it is easy to check that P_x is indeed a cone. It is reduced to 0 exactly when x is maximal in $\mathcal{B}P$. In that case one has $\|x\|_P = 1$ but notice that the converse is not true in general.

1.2 Basic definitions on PCSs

Given an at most countable set I and $u, u' \in \overline{\mathbb{R}_{\geq 0}}^I$, we set $\langle u, u' \rangle = \sum_{i \in I} u_i u'_i \in \overline{\mathbb{R}_{\geq 0}}$. Given $P \subseteq \overline{\mathbb{R}_{\geq 0}}^I$, we define $P^\perp \subseteq \overline{\mathbb{R}_{\geq 0}}^I$ as $P^\perp = \{u' \in \overline{\mathbb{R}_{\geq 0}}^I \mid \forall u \in P\ \langle u, u' \rangle \leq 1\}$. Observe that if P satisfies $\forall a \in I\, \exists x \in P\ x_a > 0$ and $\forall a \in I\, \exists m \in \mathbb{R}_{\geq 0} \forall x \in P\ x_a \leq m$ then $P^\perp \in (\mathbb{R}_{\geq 0})^I$ and $P^\perp$ satisfies the same two properties.

A probabilistic pre-coherence space (pre-PCS) is a pair $X = (|X|, \mathsf{P}X)$ where $|X|$ is an at most countable set[4] and $\mathsf{P}X \subseteq \overline{\mathbb{R}_{\geq 0}}^{|X|}$ satisfies $\mathsf{P}X^{\perp\perp} = \mathsf{P}X$. A probabilistic coherence space (PCS) is a pre-PCS X such that $\forall a \in |X|\, \exists x \in \mathsf{P}X\ x_a > 0$ and $\forall a \in |X|\, \exists m \in \mathbb{R}_{\geq 0} \forall x \in \mathsf{P}X\ x_a \leq m$ so that $\mathsf{P}X \subseteq (\mathbb{R}_{\geq 0})^{|X|}$.

Given any PCS X we can define a cone $\overline{\mathsf{P}}X$ as follows:

$$\overline{\mathsf{P}}X = \{x \in (\mathbb{R}_{\geq 0})^{|X|} \mid \exists \varepsilon > 0\ \varepsilon x \in \mathsf{P}X\}$$

that we equip with the following norm: $\|x\|_{\overline{\mathsf{P}}X} = \inf\{r > 0 \mid x \in r\,\mathsf{P}X\}$ and then it is easy to check that $\mathcal{B}(\overline{\mathsf{P}}X) = \mathsf{P}X$. We simply denote this norm as $\|_\|_X$.

Given $t \in \overline{\mathbb{R}_{\geq 0}}^{I \times J}$ considered as a matrix (where I and J are at most countable sets) and $u \in \overline{\mathbb{R}_{\geq 0}}^I$, we define $t\,u \in \overline{\mathbb{R}_{\geq 0}}^J$ by $(t\,u)_j = \sum_{i \in I} t_{i,j} u_i$ (usual formula for applying a matrix to a vector), and if $s \in \overline{\mathbb{R}_{\geq 0}}^{J \times K}$ we define the product $s\,t \in \overline{\mathbb{R}_{\geq 0}}^{I \times K}$ of the matrix s and t as usual by $(s\,t)_{i,k} = \sum_{j \in J} t_{i,j} s_{j,k}$. This is an associative operation.

Let X and Y be PCSs, a morphism from X to Y is a matrix $t \in (\mathbb{R}_{\geq 0})^{|X| \times |Y|}$ such that $\forall x \in \mathsf{P}X\ t\,x \in \mathsf{P}Y$. It is clear that the identity matrix is a morphism from X to X and that the matrix product of two morphisms is a morphism and therefore, PCS equipped with this notion of morphism form a category **Pcoh**.

The condition $t \in \mathbf{Pcoh}(X, Y)$ is equivalent to $\forall x \in \mathsf{P}X\, \forall y' \in \mathsf{P}Y^\perp\ \langle t\,x, y' \rangle \leq 1$ but $\langle t\,x, y' \rangle = \langle t, x \otimes y' \rangle$ where $(x \otimes y')_{(a,b)} = x_a y'_b$. This strongly suggests to introduce a construction $X \otimes Z$, given two PCSs X and Z, by setting $|X \otimes Z| = |X| \times |Z|$ and $\mathsf{P}(X \otimes Z) = \{x \otimes z \mid x \in \mathsf{P}X \text{ and } z \in \mathsf{P}Z\}^{\perp\perp}$ where $(x \otimes z)_{(a,c)} = x_a z_c$. Then it is easy to see that $X \otimes Z$ is not only a pre-PCS, but actually a PCS and that we have equipped in that way the category **Pcoh** with a symmetric monoidal structure for which it is $*$-autonomous wrt. a dualizing object $\perp = 1 = (\{*\}, [0,1])$ (it is at the same time the unit of $\otimes$ and $X^\perp \simeq (X \multimap \perp)$ up to a trivial iso).

The category **Pcoh** is cartesian: if $(X_i)_{i \in I}$ is an at most countable family of PCSs, then $(\&_{i \in I} X_i, (\pi_i)_{i \in I})$ is the cartesian product of the X_is, with $|\&_{i \in I} X_i| = \cup_{i \in I} \{i\} \times |X_i|$, $(\pi_i)_{(j,a),a'} = 1$ if $i = j$ and $a = a'$ and $(\pi_i)_{(j,a),a'} = 0$ otherwise, and $x \in \mathsf{P}(\&_{i \in I} X_i)$ if $\pi_i\,x \in \mathsf{P}X_i$ for each $i \in I$ (for $x \in (\mathbb{R}_{\geq 0})^{|\&_{i \in I} X_i|}$). Given $t_i \in \mathbf{Pcoh}(Y, X_i)$, the unique morphism $t = \langle t_i \rangle_{i \in I} \in \mathbf{Pcoh}(Y, \&_{i \in I} X_i)$ such that $\pi_i\,t = t_i$ is simply defined by $t_{b,(i,a)} = (t_i)_{a,b}$. The

[4] This restriction is not technically necessary, but very meaningful from a philosophic point of view; the non countable case should be handled via measurable spaces and then one has to consider more general objects as in [10] for instance.

dual operation $\oplus_{i\in I} X_i$, which is a coproduct, is characterized by $|\oplus_{i\in I} X_i| = \cup_{i\in I}\{i\} \times |X_i|$ and $x \in \mathsf{P}(\oplus_{i\in I} X_i)$ and $\sum_{i\in I} \|\pi_i\, x\|_{X_i} \leq 1$. A particular case is $\mathsf{N} = \oplus_{n\in\mathbb{N}} X_n$ where $X_n = 1$ for each n. So that $|\mathsf{N}| = \mathbb{N}$ and $x \in (\mathbb{R}_{\geq 0})^{\mathbb{N}}$ belongs to PN if $\sum_{n\in\mathbb{N}} x_n \leq 1$ (that is, x is a sub-probability distribution on $\mathbb{N}$). There are successor and predecessor morphisms $\overline{\mathsf{suc}}, \overline{\mathsf{pred}} \in \mathbf{Pcoh}(\mathsf{N},\mathsf{N})$ given by $\overline{\mathsf{suc}}_{n,n'} = \delta_{n+1,n'}$ and $\overline{\mathsf{pred}}_{n,n'} = 1$ if $n = n' = 0$ or $n = n'+1$ (and $\overline{\mathsf{pred}}_{n,n'} = 0$ in all other cases). An element of $\mathbf{Pcoh}(\mathsf{N},\mathsf{N})$ is a (sub)stochastic matrix and our model should be understood as this kind of representation of programs.

As to the exponentials, one sets $|!X| = \mathcal{M}_{\mathrm{fin}}(|X|)$ and $\mathsf{P}(!X) = \{x^! \mid x \in \mathsf{P}X\}^{\perp\perp}$ where, given $\mu \in \mathcal{M}_{\mathrm{fin}}(|X|)$, $x^!_\mu = x^\mu = \prod_{a\in|X|} x_a^{\mu(a)}$. Then given $t \in \mathbf{Pcoh}(X,Y)$, one defines $!t \in \mathbf{Pcoh}(!X,!Y)$ in such a way that $!t\, x^! = (t\, x)^!$ (the precise definition is not relevant here; it is completely determined by this equation). We do not need here to specify the monoidal comonad structure of this exponential. The resulting cartesian closed category[5] $\mathbf{Pcoh}_!$ can be seen as a category of functions (actually, of stable functions as proved in [3]). Indeed, a morphism $t \in \mathbf{Pcoh}_!(X,Y) = \mathbf{Pcoh}(!X,Y) = \mathsf{P}(!X \multimap Y)$ is completely characterized by the associated function $\widehat{t} : \mathsf{P}X \to \mathsf{P}Y$ such that $\widehat{t}(x) = t\, x^! = \left(\sum_{\mu\in|!X|} t_{\mu,b} x^\mu\right)_{b\in|Y|}$ so that we consider morphisms as power series (they are in particular monotonic and Scott continuous functions $\mathsf{P}X \to \mathsf{P}Y$). In this cartesian closed category, the product of a family $(X_i)_{i\in I}$ is $\&_{i\in I} X_i$ (written X^I if $X_i = X$ for all i), which is compatible with our viewpoint on morphisms as functions since $\mathsf{P}(\&_{i\in I} X_i) = \prod_{i\in I} \mathsf{P}X_i$ up to trivial iso. The object of morphisms from X to Y is $!X \multimap Y$ with evaluation mapping $(t,x) \in \mathsf{P}(!X \multimap Y) \times \mathsf{P}X$ to $\widehat{t}(x)$ that we simply denote as $t(x)$ from now on. The well defined function $\mathsf{P}(!X \multimap X) \to \mathsf{P}X$ which maps t to $\sup_{n\in\mathbb{N}} t^n(0)$ is a morphism of $\mathbf{Pcoh}_!$ (and thus can be described as a power series in the vector $t = (t_{m,a})_{m\in\mathcal{M}_{\mathrm{fin}}(|X|),a\in|X|})$ by standard categorical considerations using cartesian closeness: it provides us with fixed point operators at all types.

2 Probabilistic PCF, time expectation and derivatives

We introduce now the probabilistic functional programming language considered in this paper. The operational semantics is presented using elementary probability theoretic tools.

2.1 The core language

The types and terms are given by

$$\begin{aligned} \sigma,\tau,\ldots &:= \iota \mid \sigma \Rightarrow \tau \\ M,N,P\ldots &:= \underline{n} \mid \mathsf{succ}(M) \mid \mathsf{pred}(M) \mid x \mid \mathsf{coin}(r) \mid \mathsf{let}(x,M,N) \mid \mathsf{if}(M,N,P) \\ &\quad\mid (M)N \mid \lambda x^\sigma\, M \mid \mathsf{fix}(M) \end{aligned}$$

See Fig. 1 for the typing rules, with typing contexts $\Gamma = (x_1 : \sigma_1, \ldots, x_n : \sigma_n)$.

2.1.1 Denotational semantics

We survey briefly the interpretation of pPCF in PCSs thoroughly described in [10]. Types are interpreted by $[\![\iota]\!] = \mathsf{N}$ and $[\![\sigma \Rightarrow \tau]\!] = ![\![\sigma]\!] \multimap [\![\tau]\!]$. Given $M \in$ pPCF such that $\Gamma \vdash M : \sigma$ (with $\Gamma = (x_1 : \sigma_1, \ldots, x_k : \sigma_k)$) one defines $[\![M]\!]_\Gamma \in \mathbf{Pcoh}_!(\&_{i=1}^k [\![\sigma_i]\!], [\![\sigma]\!])$ (a

[5] This is the Kleisli category of "!" which has actually a comonad structure that we do not make explicit here, again we refer to [5, 10].

$$\frac{}{\Gamma \vdash \underline{n} : \iota} \qquad \frac{}{\Gamma, x : \sigma \vdash x : \sigma} \qquad \frac{\Gamma \vdash M : \iota}{\Gamma \vdash \mathsf{succ}(M) : \iota} \qquad \frac{\Gamma \vdash M : \iota}{\Gamma \vdash \mathsf{pred}(M) : \iota}$$

$$\frac{\Gamma \vdash M : \iota \qquad \Gamma \vdash N : \sigma \qquad \Gamma \vdash P : \sigma}{\Gamma \vdash \mathsf{if}(M, N, P) : \sigma} \qquad \frac{\Gamma \vdash M : \iota \qquad \Gamma, z : \iota \vdash N : \sigma}{\Gamma \vdash \mathsf{let}(z, M, N) : \sigma}$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x^\sigma\, M : \sigma \Rightarrow \tau} \qquad \frac{\Gamma \vdash M : \sigma \Rightarrow \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash (M)N : \tau} \qquad \frac{\Gamma \vdash M : \sigma \Rightarrow \sigma}{\Gamma \vdash \mathsf{fix}(M) : \sigma} \qquad \frac{r \in [0,1] \cap \mathbb{Q}}{\Gamma \vdash \mathsf{coin}(r) : \iota}$$

$$\frac{}{\iota \vdash \varepsilon} \qquad \frac{\vdash M : \sigma \qquad \tau \vdash \pi}{\sigma \Rightarrow \tau \vdash \mathsf{arg}(M) \cdot \pi} \qquad \frac{\iota \vdash \pi}{\iota \vdash \mathsf{succ} \cdot \pi} \qquad \frac{\iota \vdash \pi}{\iota \vdash \mathsf{pred} \cdot \pi}$$

$$\frac{\vdash N : \sigma \qquad \vdash P : \sigma \qquad \sigma \vdash \pi}{\iota \vdash \mathsf{if}(N, P) \cdot \pi} \qquad \frac{x : \iota \vdash N : \sigma \qquad \sigma \vdash \pi}{\iota \vdash \mathsf{let}(x, N) \cdot \pi}$$

Figure 1 Typing rules for pPCF terms and stacks.

"Kleisli morphism") that we see as a function $\prod_{i=1}^{k} \mathsf{P}[\![\sigma_i]\!] \to \mathsf{P}[\![\sigma]\!]$ as explained in Section 1.2. For instance $[\![x_i]\!]_\Gamma(\vec{u}) = u_i$, $[\![\underline{n}]\!]_\Gamma(\vec{u}) = \overline{n}$ (remember that $\overline{n} \in \mathsf{PN}$ is defined by $\overline{n}_i = \delta_{n,i}$), $[\![\mathsf{succ}(M)]\!]_\Gamma(\vec{u}) = \overline{\mathsf{suc}}\, [\![M]\!]_\Gamma(\vec{u})$ and similarly for $\mathsf{pred}(M)$, more importantly

$$[\![\mathsf{coin}(r)]\!]_\Gamma(\vec{u}) = r\,\overline{0} + (1-r)\,\overline{1} \qquad [\![\mathsf{let}(x, M, N)]\!]_\Gamma = \sum_{n \in \mathbb{N}} [\![M]\!]_\Gamma(\vec{u})_n\, [\![N\,[\underline{n}/x]]\!]_\Gamma(\vec{u})$$

$$[\![\mathsf{if}(M, N, P)]\!]_\Gamma(\vec{u}) = [\![M]\!]_\Gamma(\vec{u})_0\, [\![N]\!]_\Gamma(\vec{u}) + \left(\sum_{n \in \mathbb{N}} [\![M]\!]_\Gamma(\vec{u})_{n+1}\right) [\![P]\!]_\Gamma(\vec{u})\,.$$

Application and λ-abstraction are interpreted as usual in a cartesian closed category (in particular $[\![(M)N]\!]_\Gamma(\vec{u}) = ([\![M]\!]_\Gamma(\vec{u}))([\![N]\!]_\Gamma(\vec{u}))$. Last $[\![\mathsf{fix}(M)]\!]_\Gamma(\vec{u}) = \sup_{n \in \mathbb{N}}([\![M]\!]_\Gamma(\vec{u}))^n(0)$.

2.1.2 Operational semantics

In former papers we have presented the operational semantics of pPCF as a discrete Markov chain on states which are the closed terms of pPCF. This Markov chain implements the standard weak head reduction strategy of PCF which is deterministic for ordinary PCF but features branchings in pPCF because of the $\mathsf{coin}(r)$ construct (see [10]). Here we prefer another, though strictly equivalent, presentation of this operational semantics, based on an environment-free Krivine Machine (thus handling states which are pairs made of a closed term and a closed stack) further parameterized by an element of $\{0,1\}^{<\omega}$ to be understood as a "random tape" prescribing the values taken by the $\mathsf{coin}(r)$ terms during the execution of states. We present this machine as a partial function taking a state s, a random tape α and returning an element of $[0,1]$ to be understood as the probability that the sequence α of 0/1 choices occurs during the execution of s. We allow only execution of ground type states and accept $\underline{0}$ as the only terminating value: a completely arbitrary choice, sufficient for our purpose in this paper. Also, we insist that a terminating computation from (s, α) completely consumes the random tape α. These choices allow to fit within a completely standard discrete probability setting.

Given an extension Λ of pPCF (with the same format for typing rules), we define the associated language of stacks (called Λ-stacks).

$$\pi := \varepsilon \mid \mathsf{arg}(M) \cdot \pi \mid \mathsf{succ} \cdot \pi \mid \mathsf{pred} \cdot \pi \mid \mathsf{if}(N, P) \cdot \pi \mid \mathsf{let}(x, N) \cdot \pi$$

where M and N range over Λ. A stack typing judgment is of shape $\sigma \vdash \pi$ (meaning that it takes a term of type σ and returns an integer) and the typing rules are given in Fig. 1.

$$\mathsf{Ev}(\langle \mathsf{let}(x,M,N),\pi\rangle,\alpha) = \mathsf{Ev}(\langle M,\mathsf{let}(x,N)\cdot\pi\rangle,\alpha)$$
$$\mathsf{Ev}(\langle \underline{n},\mathsf{let}(x,N)\cdot\pi\rangle,\alpha) = \mathsf{Ev}(\langle N\,[\underline{n}/x]\,,\pi\rangle,\alpha)$$
$$\mathsf{Ev}(\langle \mathsf{if}(M,N,P),\pi\rangle) = \mathsf{Ev}(\langle M,\mathsf{if}(N,P)\cdot\pi\rangle,\alpha)$$
$$\mathsf{Ev}(\langle \underline{0},\mathsf{if}(N,P)\cdot\pi\rangle,\alpha) = \mathsf{Ev}(\langle N,\pi\rangle,\alpha)$$
$$\mathsf{Ev}(\langle \underline{n+1},\mathsf{if}(N,P)\cdot\pi\rangle,\alpha) = \mathsf{Ev}(\langle P,\pi\rangle,\alpha)$$
$$\mathsf{Ev}(\langle \lambda x^{\sigma}\,M,\mathsf{arg}(N)\cdot\pi\rangle,\alpha) = \mathsf{Ev}(\langle M\,[N/x]\,,\pi\rangle,\alpha)$$
$$\mathsf{Ev}(\langle \mathsf{fix}(M),\pi\rangle,\alpha) = \mathsf{Ev}(\langle M,\mathsf{arg}(\mathsf{fix}(M))\cdot\pi\rangle,\alpha)$$
$$\mathsf{Ev}(\langle \mathsf{coin}(r),\pi\rangle,\langle 0\rangle\alpha) = \mathsf{Ev}(\langle \underline{0},\pi\rangle,\alpha)\cdot r$$
$$\mathsf{Ev}(\langle \mathsf{coin}(r),\pi\rangle,\langle 1\rangle\alpha) = \mathsf{Ev}(\langle \underline{1},\pi\rangle,\alpha)\cdot(1-r)$$
$$\mathsf{Ev}(\langle \underline{0},\varepsilon\rangle,\langle\rangle) = 1$$

Figure 2 The pPCF Krivine Machine.

A *state* is a pair $\langle M,\pi\rangle$ (where we say that M is *in head position*) such that $\vdash M:\sigma$ and $\sigma\vdash\pi$ for some (uniquely determined) type σ, let S be the set of states. Let $\mathcal{C}_0=\{0,1\}^{<\omega}$ be the set of finite lists of booleans (random tapes), we define a *partial* function $\mathsf{Ev}:\mathsf{S}\times\mathcal{C}_0\to\mathbb{R}_{\geq 0}$ in Fig. 2. Let $\mathcal{D}(s)$ be the set of all $\alpha\in\mathcal{C}_0$ such that $\mathsf{Ev}(s,\alpha)$ is defined. When $\alpha\in\mathcal{D}(s)$, the number $\mathsf{Ev}(s,\alpha)\in[0,1]$ is the probability that the random tape α occurs during the execution. When all coins are fair (all the values of the parameters r are $1/2$), this probability is $2^{-\mathsf{len}(\alpha)}$. The sum of these (possibly infinitely many) probabilities is ≤ 1. For fitting within a standard probabilistic setting, we define a total probability distribution $\mathsf{Ev}(s):\mathcal{C}_0\to[0,1]$ as follows

$$\mathsf{Ev}(s)(\alpha)=\begin{cases}\mathsf{Ev}(s,\beta) & \text{if } \alpha=\langle 0\rangle\beta \text{ and } \beta\in\mathcal{D}(s)\\ 1-\sum_{\beta\in\mathcal{D}(s)}\mathsf{Ev}(s,\beta) & \text{if } \alpha=\langle 1\rangle\\ 0 & \text{in all other cases}\end{cases}$$

Let $\mathbb{P}_s$ be the associated probability measure[6] (we are in a discrete setting so simply $\mathbb{P}_s(A)=\sum_{\alpha\in A}\mathsf{Ev}(s)(\alpha)$ for all $A\subseteq\mathcal{C}_0$).

The event $(s\downarrow\underline{0})=\langle 0\rangle\mathcal{D}(s)$ is the set of all random tapes (up to 0-prefixing) making s reduce to $\underline{0}$. Its probability is $\mathbb{P}_s(s\downarrow\underline{0})=\sum_{\beta\in\mathcal{D}(s)}\mathsf{Ev}(s,\beta)$. In the case $s=\langle M,\varepsilon\rangle$ (with $\vdash M:\iota$) this probability is *exactly the same* as the probability of M to reduce to $\underline{0}$ in the Markov chain setting of [10] (see *e.g.* [1] for more details on the connection between these two kinds of operational semantics). So the Adequacy Theorem of [10] can be expressed as follows.

▶ **Theorem 1.** *Let $M\in\mathsf{pPCF}$ with $\vdash M:\iota$. Then $[\![M]\!]_0=\mathbb{P}_{\langle M,\varepsilon\rangle}(\langle M,\varepsilon\rangle\downarrow\underline{0})$.*

We use sometimes $\mathbb{P}(M\downarrow\underline{0})$ as an abbreviation for $\mathbb{P}_{\langle M,\varepsilon\rangle}(\langle M,\varepsilon\rangle\downarrow\underline{0})$.

2.2 Probabilistic PCF with labels and the associated random variables

In order to count the number of times a given subterm N of a closed term M of type ι is used (that is, arrives in head position) during the execution of $\langle M,\varepsilon\rangle$ in the Krivine machine of Section 2.1.2, we extend pPCF into $\mathsf{pPCF_{lab}}$ by adding a term labeling construct N^l. The typing rule for this new construct is simply $\dfrac{\Gamma\vdash N:\sigma}{\Gamma\vdash N^l:\sigma}$. Of course $\mathsf{pPCF_{lab}}$-stacks involve now such labeled terms but their syntax is not extended otherwise; let $\mathsf{S_{lab}}$ be the corresponding set of states. Then we define a partial function $\mathsf{Ev_{lab}}:\mathsf{S_{lab}}\times\mathcal{C}_0\to\mathcal{M}_{\mathrm{fin}}(\mathcal{L})$ exactly as Ev apart for the following cases,

$$\mathsf{Ev_{lab}}(\langle M^l,\pi\rangle,\alpha)=\mathsf{Ev_{lab}}(\langle M,\pi\rangle,\alpha)+[l]$$
$$\mathsf{Ev_{lab}}(\langle\mathsf{coin}(r),\pi\rangle,\langle i\rangle\alpha)=\mathsf{Ev_{lab}}(\langle\underline{i},\pi\rangle,\alpha)\qquad \mathsf{Ev_{lab}}(\langle\underline{0},\varepsilon\rangle,\langle\rangle)=0\quad\text{the empty multiset.}$$

[6] The choice of accumulating on $\langle 1\rangle$ all the complementary probability is completely arbitrary and has no impact on the result we prove because all the events of interest for us will be subsets of $\langle 0\rangle\mathcal{C}_0\subset\mathcal{C}_0$.

When applied to $\langle M, \varepsilon\rangle$, this function counts how often labeled subterms of M arrive in head position during the reduction; this number depends of course on the random tape provided as argument together with the state. The result is a finite multiset of labels.

Let $\mathcal{D}_{\mathsf{lab}}(s)$ be the set of αs such that $\mathsf{Ev}_{\mathsf{lab}}(s, \alpha)$ is defined. Defining $\underline{s} \in \mathsf{S}$ as s stripped from its labels, we clearly have $\mathcal{D}_{\mathsf{lab}}(s) = \mathcal{D}(\underline{s})$. We define a r.v.[7] $\mathsf{Ev}_{\mathsf{lab}}(s) : \mathcal{C}_0 \to \mathcal{M}_{\mathrm{fin}}(\mathcal{L})$ by

$$\mathsf{Ev}_{\mathsf{lab}}(s)(\alpha) = \begin{cases} \mathsf{Ev}_{\mathsf{lab}}(s, \beta) & \text{if } \alpha = \langle 0\rangle\beta \text{ and } \beta \in \mathcal{D}(s) \\ 0 & \text{in all other cases.} \end{cases}$$

Let $l \in \mathcal{L}$ and let $\mathsf{Ev}_{\mathsf{lab}}(s)_l : \mathcal{C}_0 \to \mathbb{N}$ be the *integer* r.v. defined by $\mathsf{Ev}_{\mathsf{lab}}(s)_l(\alpha) = \mathsf{Ev}_{\mathsf{lab}}(s)(\alpha)(l)$. Its expectation is

$$\begin{aligned} \mathbb{E}(\mathsf{Ev}_{\mathsf{lab}}(s)_l) &= \sum_{n\in\mathbb{N}} n\, \mathbb{P}_s(\mathsf{Ev}_{\mathsf{lab}}(s)_l = n) = \sum_{n\in\mathbb{N}} n \sum_{\substack{\mu\in\mathcal{M}_{\mathrm{fin}}(\mathcal{L}) \\ \mu(l)=n}} \mathbb{P}_s(\mathsf{Ev}_{\mathsf{lab}}(s) = \mu) \\ &= \sum_{\mu\in\mathcal{M}_{\mathrm{fin}}(L)} \mu(l)\mathbb{P}_s(\mathsf{Ev}_{\mathsf{lab}}(s) = \mu)\,. \end{aligned} \tag{1}$$

This is the expected number of occurrences of l-labeled subterms of s arriving in head position during successful executions of s. It is more meaningful to condition this expectation under convergence of the execution of s (that is, under the event $\underline{s} \downarrow \underline{0}$). We have $\mathbb{E}(\mathsf{Ev}_{\mathsf{lab}}(s)_l \mid \underline{s} \downarrow \underline{0}) = \mathbb{E}(\mathsf{Ev}_{\mathsf{lab}}(s)_l)/\mathbb{P}_{\underline{s}}(\underline{s} \downarrow \underline{0})$ as the r.v. $\mathsf{Ev}_{\mathsf{lab}}(s)_l$ vanishes outside the event $s \downarrow \underline{0}$ since $\mathcal{D}_{\mathsf{lab}}(s) = \mathcal{D}(\underline{s})$.

Our goal now is to extract this expectation from the denotational semantics of a term M such that $\vdash M : \iota$, which contains labeled subterms, or rather of a term suitably definable from M. The general idea is to replace in M each N^l (where N has type σ) with $\mathsf{if}(x_l, N, \Omega^\sigma)$ where $\vec{x} = (x_l)_{l\in L}$ (for some finite subset L of $\mathcal{L}$ containing all the labels occurring in M) is a family of pairwise distinct variables of type ι and $\Omega^\sigma = \mathsf{fix}(\lambda x^\sigma\, x)$. We obtain in that way a term $\mathsf{sp}_{\vec{x}}M$ whose semantics $[\![\mathsf{sp}_{\vec{x}}M]\!]_{\vec{x}}$ is an element of $\mathbf{Pcoh}_!(\mathsf{N}^L, \mathsf{N})$ that we can consider as an analytic function $(\mathsf{PN})^L \to \mathsf{PN}$ and which therefore induces an analytic function $f : [0,1]^L \to [0,1]$ by $f(\vec{r}) = [\![M']\!]((r_l\overline{0})_{l\in L})_0$ (where $\vec{r}\,\overline{0} = (r_l\,\overline{0})_{l\in L} \in \mathsf{PN}^L$ for $\vec{r} \in [0,1]^L$). Our main claim is that the expectation of the number of uses of subterms labeled by l is $\frac{\partial f(\vec{r})}{\partial r_l}(1, \dots, 1)$.

In order to reduce this problem to Theorem 1, we need a further "Krivine machine" with has as many random tapes as elements of L (plus one for the plain $\mathsf{coin}(_)$ constructs occurring in M).

2.3 Probabilistic PCF with labeled coins

Let $\mathsf{pPCF}_{\mathsf{lc}}$ be pPCF extended with a construct $\mathsf{lcoin}(l, r)$ typed as $\dfrac{r \in [0,1] \cap \mathbb{Q} \text{ and } l \in \mathcal{L}}{\Gamma \vdash \mathsf{lcoin}(l, r) : \iota}$ This language features the usual $\mathsf{coin}(r)$ construct for probabilistic choice as well as a supply of identical constructs labeled by $\mathcal{L}$ that we will use to simulate the counting of Section 2.2. Of course $\mathsf{pPCF}_{\mathsf{lc}}$-stacks involve now terms with labeled coins but their syntax is not extended otherwise; let S_{lc} be the corresponding set of states. We use $\mathsf{lab}(M)$ for the set of labels occurring in M (and similarly $\mathsf{lab}(s)$ for $s \in \mathsf{S}_{\mathsf{lc}}$). Given a *finite* subset L of $\mathcal{L}$, we use $\mathsf{pPCF}_{\mathsf{lc}}(L)$ for the set of terms M such that $\mathsf{lab}(M) \subseteq L$ and we define similarly $\mathsf{S}_{\mathsf{lc}}(L)$. We also use the similar notations $\mathsf{pPCF}_{\mathsf{lab}}(L)$ and $\mathsf{S}_{\mathsf{lab}}(L)$.

[7] That is, simply, a function since we are in a discrete probability setting.

The partial function $\mathsf{Ev}_{\mathsf{lc}} : \mathsf{S}_{\mathsf{lc}}(L) \times \mathcal{C}_0 \times \mathcal{C}_0^L \to \mathbb{R}_{\geq 0}$ is defined exactly as Ev (for the unlabeled $\mathsf{coin}(r)$, we use only the first parameter in $\mathcal{C}_0$), extended by the following rules:

$$\mathsf{Ev}_{\mathsf{lc}}(\langle \mathsf{lcoin}(l,r), \pi\rangle, \alpha, \vec{\beta}) = \begin{cases} \mathsf{Ev}_{\mathsf{lc}}(\langle \underline{0}, \pi\rangle, \alpha, \vec{\beta}\,[\gamma/l]) \cdot r & \text{if } \beta(l) = \langle 0\rangle\gamma \\ \mathsf{Ev}_{\mathsf{lc}}(\langle \underline{1}, \pi\rangle, \alpha, \vec{\beta}\,[\gamma/l]) \cdot (1-r) & \text{if } \beta(l) = \langle 1\rangle\gamma \end{cases}$$

where $\vec{\beta} = (\beta(l))_{l\in L}$ stands for an L-indexed family of elements of $\mathcal{C}_0$ and $\vec{\beta}\,[\gamma/l]$ is the family $\vec{\delta}$ such that $\delta(l') = \beta(l')$ if $l' \neq l$ and $\delta(l) = \gamma$. We define $\mathcal{D}_{\mathsf{lc}}(s) \subseteq \mathcal{C}_0 \times \mathcal{C}_0^L$ as the domain of the partial function $\mathsf{Ev}_{\mathsf{lc}}(s, _, _)$. Let $\underline{s} \in \mathsf{S}$ be obtained by stripping s from its labels (so that $\underline{\mathsf{lcoin}(l,r)} = \mathsf{coin}(r)$). And $\underline{M} \in \mathsf{pPCF}$ is defined similarly.

▶ **Lemma 2.** *For all $s \in \mathsf{S}_{\mathsf{lc}}(L)$*

$$\mathbb{P}_{\underline{s}}(\underline{s} \downarrow \underline{0}) = \sum_{(\alpha,\vec{\beta})\in\mathcal{D}_{\mathsf{lc}}(s)} \mathsf{Ev}_{\mathsf{lc}}(s, \alpha, \vec{\beta})\,.$$

Proof. (Sketch) With each $(\alpha, \vec{\beta}) \in \mathcal{D}_{\mathsf{lc}}(s)$ we can associate a uniquely defined $\eta_s(\alpha, \vec{\beta}) \in \mathcal{D}(\underline{s})$ which is a shuffle of α and of the $\beta(l)$'s (for $l \in L$) such that $\mathsf{Ev}_{\mathsf{lc}}(s, \alpha, \vec{\beta}) = \mathsf{Ev}(s, \eta_s(\alpha, \vec{\beta}))$, uniquely determined by the run of $(s, \alpha, \vec{\beta})$ in the "machine" $\mathsf{Ev}_{\mathsf{lab}}$. This mapping η_s (which is defined much like $\mathsf{Ev}_{\mathsf{lc}}(s, _, _)$) is easily seen to be bijective. ◀

2.3.1 Spying labeled terms in pPCF

We arrive to the last step, which consists in turning a *closed* labeled term M (with labels in the finite set L) into the already mentioned term $\mathsf{sp}_{\vec{x}}(M)$, defined in such a way that $[\![\underline{\mathsf{lc}_{\vec{r}}(M)}]\!]$ has a simple expression in terms of $\mathsf{sp}_{\vec{x}}(M)$ (Lemma 5), allowing to relate the coefficients of the power series interpreting $\mathsf{sp}_{\vec{x}}(M)$ in terms of probability of reduction of the machine $\mathsf{Ev}_{\mathsf{lab}}$ with given resulting multisets of labels (Equation (2)). This in turn is the key to the proof of Theorem 6.

Given $\vec{r} = (r_l)_{l\in L} \in (\mathbb{Q} \cap [0,1])^L$, we define a (type preserving) translation $\mathsf{lc}_{\vec{r}} : \mathsf{pPCF}_{\mathsf{lab}}(L) \to \mathsf{pPCF}_{\mathsf{lc}}$ by induction on terms. For all term constructs but labeled terms, the transformation does nothing (for instance $\mathsf{lc}_{\vec{r}}(x) = x$, $\mathsf{lc}_{\vec{r}}(\lambda x^\sigma\, M) = \lambda x^\sigma\, \mathsf{lc}_{\vec{r}}(M)$ etc), the only non trivial case being $\mathsf{lc}_{\vec{r}}(M^l) = \mathsf{if}(\mathsf{lcoin}(l, r_l), \mathsf{lc}_{\vec{r}}(M), \Omega^\sigma)$ where σ is the type[8] of M.

▶ **Lemma 3.** *Let $s \in \mathsf{S}_{\mathsf{lab}}(L)$. Then $\mathcal{D}_{\mathsf{lab}}(s) = \mathcal{D}(\underline{s})$, $\mathcal{D}_{\mathsf{lc}}(\mathsf{lc}_{\vec{r}}(s)) = \{(\alpha, (\langle 0\rangle^{\mathsf{Ev}_{\mathsf{lab}}(s,\alpha)(l)})_{l\in L}) \mid \alpha \in \mathcal{D}(\underline{s})\}$ and $\mathsf{Ev}_{\mathsf{lc}}(\mathsf{lc}_{\vec{r}}(s), \alpha, \langle 0\rangle^{\mathsf{Ev}_{\mathsf{lab}}(s,\alpha)(l)}) = \mathbb{P}_{\underline{s}}(\{\langle 0\rangle\alpha\})(\vec{r})^{\mathsf{Ev}_{\mathsf{lab}}(s,\alpha)}$.*

Of course $\langle 0\rangle^n$ stands for the sequence $\langle 0, \ldots, 0\rangle$ (with n occurrences of 0). The proof is by induction on the length of α and boils down to the observation that $\mathcal{D}(\langle \Omega^\sigma, \pi\rangle) = \emptyset$ for any (well typed) stack π. Remember that $\mathbb{P}_{\underline{s}}(\{\langle 0\rangle\alpha\}) = \mathsf{Ev}(\underline{s}, \alpha)$ and that $(\vec{r})^\mu = \prod_{l\in l} r_l^{\mu(l)}$ for all $\mu \in \mathcal{M}_{\mathrm{fin}}(L)$.

We consider a last type preserving translation from $\mathsf{pPCF}_{\mathsf{lab}}(L)$ to pPCF: let $\vec{x}$ be a L-indexed family of pairwise distinct variables (that we identify with the typing context $(x_l : \iota)_{l\in L}$). If $M \in \mathsf{pPCF}_{\mathsf{lab}}(L)$ with $\Gamma \vdash M : \sigma$ (assuming that no free variable of M occurs in $\vec{x}$) we define $\mathsf{sp}_{\vec{x}}(M)$ with $\Gamma, \vec{x} \vdash \mathsf{sp}_{\vec{x}}(M) : \sigma$ by induction on M. The unique non trivial case is $\mathsf{sp}_{\vec{x}}(M^l) = \mathsf{if}(x_l, \mathsf{sp}_{\vec{x}}(M), \Omega^\sigma)$ where σ is the type of M.

[8] *A priori* this type is known only if we know the type of the free variables of M, so to be more precise this translation should be specified in a given typing context; this can easily be fixed by adding a further parameter to lc at the price of heavier notations.

▶ **Lemma 4.** *Let $M \in \mathsf{pPCF_{lab}}(L)$ with $\vdash M : \sigma$. If $\vec{\rho} \in \mathcal{M}_{\mathrm{fin}}(\mathbb{N})^L = \mathcal{M}_{\mathrm{fin}}(L \times \mathbb{N})$ and $a \in |[\![\sigma]\!]|$ satisfy $([\![\mathsf{sp}_{\vec{x}}(M)]\!]_{\vec{x}})_{(\vec{\rho},a)} \neq 0$ then $\rho_l(n) \neq 0 \Rightarrow n = 0$.*

The proof is a simple induction on M (of course we also have to consider open terms) and uses the fact that $[\![\Omega^\sigma]\!] = 0$.

Given $\mu \in \mathcal{M}_{\mathrm{fin}}(L)$, we use $\mu\,[0]$ for the element ρ of $\mathcal{M}_{\mathrm{fin}}(\mathbb{N})^L$ such that $\rho_l(n) = \mu(l)$ if $n = 0$ and $\rho_l(n) = 0$ otherwise.

▶ **Lemma 5.** *Let $\vec{r} \in (\mathbb{Q} \cap [0,1])^L$ and $M \in \mathsf{pPCF_{lab}}(L)$ with $\vdash M : \sigma$. Then $[\![\mathsf{sp}_{\vec{x}}(M)]\!]_{\vec{x}}(\vec{r}\,\overline{0}) = [\![\underline{\mathsf{lc}_{\vec{r}}(M)}]\!]$.*

Easy induction on M based on the fact that $[\![\mathsf{coin}(r)]\!] = r\overline{0} + (1-r)\overline{1}$ (again, one needs a more general statement involving open terms).

By Lemma 5, $[\![\underline{\mathsf{lc}_{\vec{r}}(M)}]\!]_0 = \sum_{\mu \in \mathcal{M}_{\mathrm{fin}}(L)} ([\![\mathsf{sp}_{\vec{x}}(M)]\!]_{\vec{x}})_{(\mu\,[0],0)} (\vec{r})^\mu$. By Theorem 1, we have

$$\begin{aligned}
[\![\underline{\mathsf{lc}_{\vec{r}}(M)}]\!]_0 &= \mathbb{P}_{\underline{\mathsf{lc}_{\vec{r}}(\langle M,\varepsilon\rangle)}}(\underline{\mathsf{lc}_{\vec{r}}(\langle M,\varepsilon\rangle)} \downarrow \underline{0}) \\
&= \sum_{(\alpha,\vec{\beta}) \in \mathcal{D}_{\mathsf{lc}}(\mathsf{lc}_{\vec{r}}(\langle M,\varepsilon\rangle))} \mathsf{Ev}_{\mathsf{lc}}(\mathsf{lc}_{\vec{r}}(\langle M,\varepsilon\rangle), \alpha, \vec{\beta}) \quad \text{by Lemma 2} \\
&= \sum_{\alpha \in \mathcal{D}(\underline{\langle M,\varepsilon\rangle})} \mathsf{Ev}(\underline{\langle M,\varepsilon\rangle}, \alpha) \prod_{l \in L} r_l^{\mathsf{Ev_{lab}}(\langle M,\varepsilon\rangle,\alpha)(l)} \quad \text{by Lemma 3} \\
&= \sum_{\mu \in \mathcal{M}_{\mathrm{fin}}(L)} \left(\sum_{\substack{\alpha \in \langle 0 \rangle \mathcal{C}_0 \\ \mathsf{Ev_{lab}}(\langle M,\varepsilon\rangle)(\alpha) = \mu}} \mathsf{Ev}(\underline{\langle M,\varepsilon\rangle})(\alpha) \right) (\vec{r})^\mu
\end{aligned}$$

and since this holds for all $\vec{r} \in (\mathbb{Q} \cap [0,1])^L$, we must have, for all $\mu \in \mathcal{M}_{\mathrm{fin}}(L)$,

$$([\![\mathsf{sp}_{\vec{x}}(M)]\!]_{\vec{x}})_{(\mu\,[0],0)} = \sum_{\substack{\alpha \in \langle 0 \rangle \mathcal{C}_0 \\ \mathsf{Ev_{lab}}(\langle M,\varepsilon\rangle)(\alpha) = \mu}} \mathsf{Ev}(\underline{\langle M,\varepsilon\rangle})(\alpha) = \mathbb{P}_{\underline{\langle M,\varepsilon\rangle}}(\mathsf{Ev_{lab}}(\langle M,\varepsilon\rangle) = \mu) \tag{2}$$

Let $l \in L$, we have

$$\begin{aligned}
\mathbb{E}(\mathsf{Ev_{lab}}(\langle M,\varepsilon\rangle)_l) &= \sum_{\mu \in \mathcal{M}_{\mathrm{fin}}(L)} \mu(l) \mathbb{P}_{\langle M,\varepsilon\rangle}(\mathsf{Ev_{lab}}(\langle M,\varepsilon\rangle) = \mu) \quad \text{by Equation (1)} \\
&= \sum_{\mu \in \mathcal{M}_{\mathrm{fin}}(L)} \mu(l) [\![\mathsf{sp}_{\vec{x}}(M)]\!]_{\vec{x}})_{(\mu\,[0],0)} \quad \text{by Equation (2)} \\
&= \frac{\partial [\![\mathsf{sp}_{\vec{x}} M]\!]_{\vec{x}}(\vec{r}\,\overline{0})_0}{\partial r_l}(1,\dots,1)\,.
\end{aligned}$$

Indeed, given $\vec{r} \in [0,1]^L$ one has $[\![\mathsf{sp}_{\vec{x}} M]\!]_{\vec{x}}(\vec{r}\,\overline{0})_0 = \sum_{\mu \in \mathcal{M}_{\mathrm{fin}}(L)} [\![\mathsf{sp}_{\vec{x}}(M)]\!]_{\vec{x}})_{(\mu\,[0],0)} \vec{r}^\mu$ and $\frac{\partial \vec{r}^\mu}{\partial r_l}(1,...,1) = \mu(l)$, whence the last equation.

▶ **Theorem 6.** *Let $M \in \mathsf{pPCF_{lab}}(L)$ with $\vdash M : \iota$. Then*

$$\mathbb{E}(\mathsf{Ev_{lab}}(\langle M,\varepsilon\rangle)_l \mid \langle \underline{M},\varepsilon\rangle \downarrow \underline{0}) = \frac{\partial [\![\mathsf{sp}_{\vec{x}} M]\!](\vec{r}\,\overline{0})}{\partial r_l}(1,\dots,1)/[\![\underline{M}]\!]_0\,.$$

▶ **Example 7.** The point of this formula is that we can apply it to algebraic expressions of the semantics of the program. Consider the following term M_q (for $q \in \mathbb{Q} \cap [0,1]$) such that $\vdash M_q : \iota \Rightarrow \iota$: $M_q = \mathsf{fix}(\lambda f^{\iota \Rightarrow \iota}\, \lambda x^\iota\, \mathsf{if}(\mathsf{coin}(q), \mathsf{if}((f)x, \mathsf{if}((f)x, \underline{0}, \Omega^\iota), \Omega^\iota), \mathsf{if}(x, \mathsf{if}(x, \underline{0}, \Omega^\iota), \Omega^\iota)))$, we study $(M_q)\underline{0}^l$ (for a fixed label $l \in \mathcal{L}$). So in this example, "time" means "number of uses

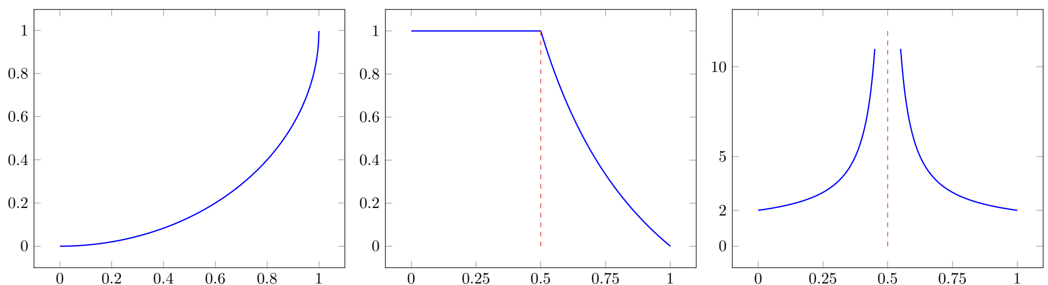


Figure 3 Plot of $\varphi_{0.5}(u)$ with u on the x-axis (vertical slope at $u = 1$). Plots of $\varphi_q(1)$ and $\mathbb{E}(\mathsf{Ev}_{\mathsf{lab}}(\langle(M_q)\underline{0}^l, \varepsilon\rangle)_l \mid \langle(M_q)\underline{0}, \varepsilon\rangle \downarrow \underline{0})$ with q on the x-axis. See Example 7.

of the parameter $\underline{0}$". For all $v \in \mathsf{PN}$, we have $[\![M_q]\!](v) = \varphi_q(v_0)\,\overline{0}$ where $\varphi_q : [0,1] \to [0,1]$ is such that $\varphi_q(u)$ is the least element of $[0,1]$ which satisfies $\varphi_q(u) = (1-q)\,u^2 + q\,\varphi_q(u)^2$. So $\varphi_q(u) = (1 - \sqrt{1-4q(1-q)u^2})/2q$ if $q > 0$ and $\varphi_0(u) = u^2$, the choice between the two solutions of the quadratic equation being determined by the fact that the resulting function φ_q must be monotonic in u. So by Theorem 1 (for $q \in (0,1]$)

$$\mathbb{P}((M_q)\underline{0} \downarrow \underline{0}) = \varphi_q(1) = \frac{1-|2q-1|}{2q} = \begin{cases} 1 & \text{if } q \leq 1/2 \\ \frac{1-q}{q} & \text{if } q > 1/2\,. \end{cases} \tag{3}$$

Observe that we have also $\mathbb{P}(M_0 \downarrow \underline{0}) = \varphi_0(1) = 1$ so that Equation (3) holds for all $q \in [0,1]$ (the corresponding curve is the second one in Fig. 3). Then by Theorem 6 we have $\mathbb{E}(\mathsf{Ev}_{\mathsf{lab}}(\langle(M_q)\underline{0}^l, \varepsilon\rangle)_l \mid \langle(M_q)\underline{0}, \varepsilon\rangle \downarrow \underline{0}) = \varphi_q'(1)/\varphi_q(1)$. Since $\varphi_q(u) = (1-q)\,u^2 + q\,\varphi_q(u)^2$ we have $\varphi_q'(u) = 2(1-q)u + 2q\varphi_q'(u)\varphi_q(u)$ and hence $\varphi_q'(1) = 2(1-q)/(1-2q\varphi_q(1))$, so that $\varphi_q'(1)/\varphi_q(1) = 2(1-q)/(1-2q)$ if $q < 1/2$, $\varphi_{1/2}'(1)/\varphi_q(1) = \infty$ and $\varphi_q'(1)/\varphi_q(1) = 2(1-q)/(2q-1)$ if $q > 1/2$ (using the expression of $\varphi_q(1)$ given by Equation (3)), see the third curve in Fig. 3. For $q > 1/2$ notice that the conditional time expectation *and* the probability of convergence decrease when q tends to 1. When q is very close to 1, $(M_q)\underline{0}$ has a very low probability to terminate, but when it does, it uses its argument only twice. For $q = 1/2$ we have almost sure termination with an infinite expected computation time.

3 Differentials and distances

3.1 Order theoretic characterization of PCSs

The following simple lemma will prove quite useful in the sequel. It is proven in [12] in a rather sketchy way, we provide here a detailed proof for further references. We say that a partially ordered set S is ω-complete if any increasing sequence of elements of S has a least upper bound.

▶ **Lemma 8.** *Let I be a countable set and let $P \subseteq (\mathbb{R}_{\geq 0})^I$. Then (I,P) is a probabilistic coherence space iff the following properties hold (equipping P with the product order).*
1. *P is downwards closed and closed under barycentric combinations*
2. *P is ω-complete*
3. *and for all $a \in I$ there is $\varepsilon > 0$ such that $\varepsilon e_a \in P$ and $P_a \subseteq [0, 1/\varepsilon]$.*

Proof. The $\Rightarrow$ implication is easy (see [5]), we prove the converse, which uses the Hahn-Banach theorem in finite dimension. Let $y \in (\mathbb{R}_{\geq 0})^I$ such that $y \notin P$. We must prove that there exists $x' \in P^\perp$ such that $\langle y, x'\rangle > 1$ and $\forall x \in P\,\langle x, x'\rangle \leq 1$. Given $J \subseteq I$ and $z \in (\mathbb{R}_{\geq 0})^I$, let $z|_J$ be the element of $(\mathbb{R}_{\geq 0})^I$ which takes value z_j for $j \in J$ and 0 for $j \notin J$.

Then y is the lub of the increasing sequence $\{y|_{\{i_1,\dots,i_n\}} \mid n \in \mathbb{N}\}$ (where $i_1, i_2, \dots$ is any enumeration of I) and hence there must be some $n \in \mathbb{N}$ such that $y|_{\{i_1,\dots,i_n\}} \notin P$. Therefore it suffices to prove the result for I finite, what we assume now. Let $Q = \{x \in \mathbb{R}^I \mid (|x_i|)_{i\in I} \in P\}$ which is a convex subset of $\mathbb{R}^I$. Let $t_0 = \sup\{t \in \mathbb{R}_{\geq 0} \mid ty \in P\}$. By our closeness assumption on P, we have $t_0 y \in P$ and therefore $t_0 < 1$. Let $h : \mathbb{R}y \to \mathbb{R}$ be defined by $h(ty) = t/t_0$ ($t_0 \neq 0$ by our assumption (3) about P and because I is finite). Let $q : \mathbb{R}^I \to \mathbb{R}_{\geq 0}$ be the gauge of Q, which is the semi-norm given by $q(z) = \inf\{\varepsilon > 0 \mid z \in \varepsilon Q\}$. It is actually a norm by our assumptions on P. Observe that $h(z) \leq q(z)$ for all $z \in \mathbb{R}y$: this boils down to showing that $t \leq t_0 q(ty) = |t|\, t_0 q(y)$ for all $t \in \mathbb{R}$ which is clear since $t_0 q(y) = 1$ by definition of these numbers. Hence, by the Hahn-Banach Theorem, there exists a linear $l : \mathbb{R}^I \to \mathbb{R}$ which is $\leq q$ and coincides with h on $\mathbb{R}y$. Let $y' \in \mathbb{R}^I$ be such that $\langle z, y'\rangle = l(z)$ for all $z \in \mathbb{R}^I$ (using again the finiteness of I). Let $x' \in (\mathbb{R}_{\geq 0})^I$ be defined by $x'_i = |y'_i|$. It is clear that $\langle y, x'\rangle > 1$: since $y \in (\mathbb{R}_{\geq 0})^I$ we have $\langle y, x'\rangle \geq \langle y, y'\rangle = l(y) = h(y) = 1/t_0 > 1$. Let $N = \{i \in I \mid y'_i < 0\}$. Given $z \in P$, let $\bar{z} \in \mathbb{R}^I$ be given by $\bar{z}_i = -z_i$ if $i \in N$ and $\bar{z}_i = z_i$ otherwise. Then $\langle z, x'\rangle = \langle \bar{z}, y'\rangle = l(\bar{z}) \leq 1$ since $\bar{z} \in Q$ (by definition of Q and because $z \in P$). It follows that $x' \in P^\perp$. ◀

3.2 Local PCS and derivatives

Let X be a PCS and let $x \in \mathsf{P}X$. We define a new PCS X_x as follows. First we set $|X_x| = \{a \in |X| \mid \exists \varepsilon > 0\ x + \varepsilon e_a \in \mathsf{P}X\}$ and then $\mathsf{P}(X_x) = \{u \in (\mathbb{R}_{\geq 0})^{|X_x|} \mid x + u \in \mathsf{P}X\}$. There is a slight abuse of notation here: u is not an element of $(\mathbb{R}_{\geq 0})^{|X|}$, but we consider it as such by simply extending it with 0 values to the elements of $|X| \setminus |X_x|$. Observe also that, given $u \in \mathsf{P}X$, if $x + u \in \mathsf{P}X$, then we *must have* $u \in \mathsf{P}(X_x)$, in the sense that u necessarily vanishes outside $|X_x|$. It is clear that $(|X_x|, \mathsf{P}(X_x))$ satisfies the conditions of Lemma 8 and therefore X_x is actually a PCS, called the *local PCS of X at x*.

Let $t \in \mathbf{Pcoh}_!(X, Y)$ and let $x \in \mathsf{P}X$. Given $u \in \mathsf{P}(X_x)$, we know that $x + u \in \mathsf{P}X$ and hence we can compute $t(x + u) \in \mathsf{P}Y$: $t(x+u)_b = \sum_{\mu\in|!X|} t_{\mu,b}(x+u)^\mu = \sum_{\mu\in|!X|} t_{\mu,b} \sum_{\nu\leq\mu} \binom{\mu}{\nu} x^{\mu-\nu} u^\nu$. Upon considering only the u-constant and the u-linear parts of this summation (and remembering that actually $u \in \mathsf{P}(X_x)$), we get $t(x) + \sum_{a\in|X|} u_a \sum_{\mu\in|!X|} (\mu(a)+1) t_{\mu+[a],b} x^\mu \leq t(x+u) \in \mathsf{P}Y$. Given $a \in |X_x|$ and $b \in |Y_{t(x)}|$, we set $t'(x)_{a,b} = \sum_{\mu\in|!X|} (\mu(a)+1) t_{\mu+[a],b} x^\mu$ and we have proven that actually $t'(x) \in \mathsf{P}(X_x, Y_{t(x)})$. By definition, this linear morphism $t'(x)$ is the *derivative (or differential, or Jacobian) of t at x*[9]. It is uniquely characterized by the fact that, for all $x \in \mathsf{P}X$ and $u \in \mathsf{P}X_x$, we have

$$t(x+u) = t(x) + t'(x)\,u + \widetilde{t}(x,u) \tag{4}$$

where $\widetilde{t}$ is a power series in x and u whose all terms have global degree ≥ 2 in u.

As a typical example, consider the case where $Y = !X$ and $t = \delta = \mathsf{Id}_{!X} \in \mathbf{Pcoh}_!(X, !X)$, so that $\delta(x) = x^!$. Given $a \in |X_x|$ and $\nu \in [!X_{x^!}]$, we have

$$\delta'(x)_{a,\nu} = \sum_{\mu\in|!X|} (\mu(a)+1)\delta_{\mu+[a],\nu} x^\mu = \begin{cases} 0 & \text{if } \nu(a) = 0 \\ \nu(a) x^{\nu-[a]} & \text{if } \nu(a) > 0\,. \end{cases}$$

We know that $\delta'(x) \in \mathsf{P}(X_x \multimap !X_{x^!})$ so that $\delta'(x)$ is a "local version" of DiLL's codereliction [9]. Observe for instance that $\delta'(0)$ satisfies $\delta'(0)_{a,\nu} = \delta_{\nu,[a]}$ and therefore coincides with the ordinary definition of codereliction.

[9] But unlike our models of Differential LL, this derivative is only defined locally; this is slightly reminiscent of what happens in differential geometry.

▶ **Proposition 9** (Chain Rule). *Let $s \in \mathbf{Pcoh}_!(X,Y)$ and $t \in \mathbf{Pcoh}_!(Y,Z)$. Let $x \in \mathsf{P}X$ and $u \in \mathsf{P}X_x$. Then we have $(t \circ s)'(x)\,u = t'(s(x))\,s'(x)\,u$.*

Proof. It suffices to write

$$\begin{aligned}(t \circ s)(x+u) &= t(s(x+u)) = t(s(x) + s'(x)\,u + \widetilde{s}(x,u)) \\ &= t(s(x)) + t'(s(x))\,(s'(x)\,u + \widetilde{s}(x,u))) + \widetilde{t}(s(x), s'(x)\,u + \widetilde{s}(x,u)) \\ &= t(s(x)) + t'(s(x))\,(s'(x)\,u) + t'(s(x))\,(\widetilde{s}(x,u)) + \widetilde{t}(s(x), s'(x)\,u + \widetilde{s}(x,u))\end{aligned}$$

by linearity of $t'(s(x))$ which proves our contention by the observation that, in the power series $t'(s(x))\,(\widetilde{s}(x,u)) + \widetilde{t}(s(x), s'(x)\,u + \widetilde{s}(x,u))$, u appears with global degree ≥ 2 by what we know on $\widetilde{s}$ and $\widetilde{t}$. ◀

3.3 Glb's, lub's and distance

Since we are working with probabilistic coherence spaces, we could deal directly with families of real numbers and define these operations more concretely. We prefer not to do so to have a more canonical presentation which can be generalized to cones such as those considered in [10].

Given $x, y \in \mathsf{P}X$, observe that $x \wedge y \in \mathsf{P}X$, where $(x \wedge y)_a = \min(x_a, y_a)$, and that $x \wedge y$ is the glb of x and y in $\mathsf{P}X$ (with its standard ordering). It follows that x and y have also a lub $x \vee y \in \overline{\mathsf{P}}X$ which is given by $x \vee y = x + y - (x \wedge y)$ (and of course $(x \vee y)_a = \max(x_a, y_a)$).

Let us prove that $x + y - (x \wedge y)$ is actually the lub of x and y. First, $x \leq x + y - (x \wedge y)$ simply because $x \wedge y \leq y$. Next, let $z \in \overline{\mathsf{P}}X$ be such that $x \leq z$ and $y \leq z$. We must prove that $x + y - (x \wedge y) \leq z$, that is $x + y \leq z + (x \wedge y) = (z + x) \wedge (z + y)$, which is clear since $x + y \leq z + x, z + y$. We have used the fact that $+$ distributes over $\wedge$ so let us prove this last fairly standard property: $z + (x \wedge y) = (z + x) \wedge (z + y)$. The "$\leq$" inequation is obvious (monotonicity of $+$) so let us prove the converse, which amounts to $x \wedge y \geq (z+x) \wedge (z+y) - z$ (observe that indeed that $z \leq (z + x) \wedge (z + y)$). This in turn boils down to proving that $x \geq (z + x) \wedge (z + y) - z$ (and similarly for y) which results from $x + z \geq (z + x) \wedge (z + y)$ and we are done.

We define the distance between x and y by $\mathsf{d}_X(x,y) = \|x - (x \wedge y)\|_X + \|y - (x \wedge y)\|_X$. The only non obvious fact to check for proving that this is actually a distance is the triangular inequality, so let $x, y, z \in \mathsf{P}X$. We have $x - (x \wedge z) \leq x - (x \wedge y \wedge z) = x - (x \wedge y) + (x \wedge y) - (x \wedge y \wedge z)$ and hence $\|x - (x \wedge z)\|_X \leq \|x - (x \wedge y)\|_X + \|(x \wedge y) - (x \wedge y \wedge z)\|_X$. Now we have $(x \wedge y) \vee (y \wedge z) \leq y$, that is $(x \wedge y) + (y \wedge z) - (x \wedge y \wedge z) \leq y$, that is $(x \wedge y) - (x \wedge y \wedge z) \leq y - (y \wedge z)$. It follows that $\|x - (x \wedge z)\|_X \leq \|x - (x \wedge y)\|_X + \|y - (y \wedge z)\|_X$ and symmetrically $\|z - (x \wedge z)\|_X \leq \|z - (z \wedge y)\|_X + \|y - (y \wedge x)\|_X$ and summing up we get, as expected $\mathsf{d}_X(x,z) \leq \mathsf{d}_X(x,y) + \mathsf{d}_X(y,z)$.

3.4 A Lipschitz property

First of all, observe that, if $w \in \overline{\mathsf{P}}(X \multimap Y)$ and $x \in \overline{\mathsf{P}}X$, we have $\|w\,x\|_Y \leq \|w\|_{X \multimap Y}\,\|x\|_X$. Indeed $\frac{w}{\|w\|_{X \multimap Y}} \in \mathsf{P}(X \multimap Y)$ and $\frac{x}{\|x\|_X} \in \mathsf{P}X$, therefore $\frac{w}{\|w\|_{X \multimap Y}}\,\frac{x}{\|x\|_X} \in \mathsf{P}Y$ and our contention follows.

Let $p \in [0,1)$. If $x \in \mathsf{P}X$ and $\|x\|_X \leq p$, observe that, for any $u \in \mathsf{P}X$, one has $\|x + (1-p)u\|_X \leq \|x\|_X + (1-p)\|u\|_X \leq 1$ and hence $(1-p)u \in \mathsf{P}(X_x)$. Therefore, given $w \in \mathsf{P}(X_x \multimap Y)$, we have $\|w\,(1-p)u\|_Y \leq 1$ for all $u \in \mathsf{P}X$ and hence $(1-p)w \in \mathsf{P}(X \multimap Y)$.

Let $t \in \mathsf{P}(!X \multimap 1)$. We have seen that, for all $x \in \mathsf{P}X$ we have $t'(x) \in \mathsf{P}(X_x \multimap 1_{t(x)}) \subseteq \mathsf{P}(X_x \multimap 1)$. Therefore, if we assume that $\|x\|_X \leq p$, we have

$$(1-p)t'(x) \in \mathsf{P}(X \multimap 1) = \mathsf{P}X^{\perp} \,. \tag{5}$$

Let $x \leq y \in \mathsf{P}X$ be such that $\|y\|_X \leq p$. Observe that $2-p > 1$ and that $x+(2-p)(y-x) = y + (1-p)(y-x) \in \mathsf{P}X$ (because $\|y\|_X \leq p$ and $y - x \in \mathsf{P}X$). We consider the function $h : [0, 2-p] \to [0, 1]$ defined by $h(\theta) = t(x + \theta(y-x))$, which is clearly analytic on $[0, 2-p)$. More precisely, one has $h(\theta) = \sum_{n=0}^{\infty} c_n \theta^n$ for some sequence of non-negative real numbers c_n such that $\sum_{n=0}^{\infty} c_n (2-p)^n \leq 1$.

Therefore the derivative of h is well defined on $[0,1] \subset [0, 2-p)$ and one has $h'(\theta) = t'(x+\theta(y-x))\,(y-x) \leq \frac{\|y-x\|_X}{1-p}$ by (5), using Proposition 9. We have

$$0 \leq t(y) - t(x) = h(1) - h(0) = \int_0^1 h'(\theta)\, d\theta \leq \frac{\|y-x\|_X}{1-p}\,. \tag{6}$$

Let now $x, y \in \mathsf{P}X$ be such that $\|x\|_X, \|y\|_X \leq p$ (we don't assume any more that they are comparable). We have $|t(x) - t(y)| = |t(x) - t(x \wedge y) + t(x \wedge y) - t(y)| \leq |t(x) - t(x \wedge y)| + |t(y) - t(x \wedge y)| \leq \frac{1}{1-p}(\|x - (x \wedge y)\|_X + \|y - (x \wedge y)\|_X) = \frac{\mathsf{d}_X(x,y)}{1-p}$ by (6) since $x \wedge y \leq x, y$.

▶ **Theorem 10.** *Let $t \in \mathsf{P}(!X \multimap 1)$. Given $p \in [0,1)$, the function t is Lipschitz with Lipschitz constant $\frac{1}{1-p}$ on $\{x \in \mathsf{P}X \mid \|x\|_X \leq p\}$ when $\mathsf{P}X$ is equipped with the distance d_X, that is*

$$\forall x, y \in \mathsf{P}X \quad \|x\|_X, \|y\|_X \leq p \Rightarrow |t(x) - t(y)| \leq \frac{\mathsf{d}_X(x,y)}{1-p}\,.$$

4 Application to the observational distance in pPCF

Given a term M such that $\vdash M : \iota$, remember that we use $\mathbb{P}(M \downarrow \underline{0})$ for the probability of M to reduce to $\underline{0}$ in the probabilistic reduction system of [10], so that $\mathbb{P}(M \downarrow \underline{0}) = \mathbb{P}_{\langle M, \varepsilon\rangle}(\langle M, \varepsilon\rangle \downarrow \underline{0})$ with the (admittedly heavy) notations of Section 2. Remember that $\mathbb{P}(M \downarrow \underline{0}) = [\![M]\!]_0$ by the Adequacy Theorem of [10].

Given a type σ and two pPCF terms M, M' such that $\vdash M : \sigma$ and $\vdash M' : \sigma$, we define the *observational distance* $\mathsf{d_{obs}}(M, M')$ between M and M' as the sup of all the $|\mathbb{P}((C)M \downarrow \underline{0}) - \mathbb{P}((C)M' \downarrow \underline{0})|$ taken over terms C such that $\vdash C : \iota$ (testing contexts).

If $\varepsilon \in [0,1] \cap \mathbb{Q}$ we have $\mathsf{d_{obs}}(\mathsf{coin}(0), \mathsf{coin}(\varepsilon)) = 1$ as soon as $\varepsilon > 0$. It suffices indeed to consider the context $C = \mathsf{fix}\, f^{\iota \Rightarrow \iota}\, \lambda x^{\iota}\, \mathsf{if}(x, (f)x, z \cdot \underline{0})$. The semantics $[\![C]\!] \in \mathsf{P}(!\mathsf{N} \multimap \mathsf{N})$ is a function $c : \mathsf{PN} \to \mathsf{PN}$ such that $\forall u \in \mathsf{PN}\ c(u) = u_0 c(u) + (\sum_{i=1}^{\infty} u_i)\overline{0}$ and which is minimal (for the order relation of $\mathsf{P}(!\mathsf{N} \multimap \mathsf{N})$). If follows that

$$c(u) = \begin{cases} 0 & \text{if } u_0 = 1 \\ \frac{1}{1-u_0} \sum_{i=1}^{\infty} u_i & \text{otherwise}\,. \end{cases}$$

Then $c((1-\varepsilon)\overline{0} + \varepsilon\overline{1}) = 0$ if $\varepsilon = 0$ and $c((1-\varepsilon)\overline{0} + \varepsilon\overline{1}) = 1$ is $\varepsilon > 0$. This is a well known phenomenon called "probability amplification" in stochastic programming.

Nevertheless, we can control a tamed version of the observational distance. Given a closed pPCF term C such that $\vdash C : \sigma \Rightarrow \iota$ we define $C^{\langle p\rangle} = \lambda z^{\sigma}\, (C)\mathsf{if}(\mathsf{coin}(p), z, \Omega^{\sigma})$ and a tamed version of the observational distance is defined by

$$\mathsf{d}_{\mathsf{obs}}^{\langle p\rangle}(M, M') = \sup\left\{\left|\mathbb{P}((C^{\langle p\rangle})M \downarrow \underline{0}) - \mathbb{P}((C^{\langle p\rangle})M' \downarrow \underline{0})\right| \mid \vdash C : \sigma \Rightarrow \iota\right\}.$$

▶ **Theorem 11.** *Let $p \in [0,1) \cap \mathbb{Q}$. Let M and M' be terms such that $\vdash M : \sigma$ and $\vdash M' : \sigma$. Then we have*

$$\mathsf{d}_{\mathsf{obs}}^{\langle p\rangle}(M, M') \leq \frac{p}{1-p}\, \mathsf{d}_{[\![\sigma]\!]}([\![M]\!], [\![M']\!])\,.$$

Proof.

$$\begin{aligned}\mathsf{d}_{\mathsf{obs}}^{\langle p\rangle}(M,M') &= \sup\{|[\![C]\!](p[\![M]\!])_0 - [\![C]\!](p[\![M']\!])_0| \mid\ \vdash C:\sigma\Rightarrow\iota\}\\ &\leq \sup\{|t(p[\![M]\!]) - t(p[\![M']\!]|) \mid t\in\mathsf{P}(![\![\sigma]\!]\multimap 1)\}\\ &\leq \frac{\mathsf{d}_{[\![\sigma]\!]}(p[\![M]\!],p[\![M']\!])}{1-p} = \frac{p}{1-p}\,\mathsf{d}_{[\![\sigma]\!]}([\![M]\!],[\![M']\!])\,.\end{aligned}$$

by the Adequacy Theorem and by Theorem 10. ◀

Since $p/(1-p) = p+p^2+\cdots$ and $\mathsf{d}_{[\![\sigma]\!]}(_,_)$ is an over-approximation of the observational distance restricted to linear contexts, this inequation carries a rather clear operational intuition in terms of execution in a Krivine machine as in Section 2.1.2 (thanks to Paul-André Melliès for this observation). Indeed, using the stacks of Section 2.1.2, a *linear* observational distance on pPCF terms can easily be defined as follows, given terms M and M' such that $\vdash M:\sigma$ and $\vdash M':\sigma$:

$$\mathsf{d}_{\mathsf{lin}}(M,M') = \sup_{\sigma\vdash\pi}\left|\mathbb{P}_{\langle M,\pi\rangle}(\langle M,\pi\rangle\downarrow\underline{0}) - \mathbb{P}_{\langle M',\pi\rangle}(\langle M',\pi\rangle\downarrow\underline{0})\right|\,.$$

In view of Theorem 11 and of the fact that $\mathsf{d}_{\mathsf{lin}}(M,M')\leq\mathsf{d}_{[\![\sigma]\!]}([\![M]\!],[\![M']\!])$ (easy to prove, since each stack can be interpreted as a linear morphism in **Pcoh**), a natural and purely syntactic conjecture seems to be

$$\mathsf{d}_{\mathsf{obs}}^{\langle p\rangle}(M,M') \leq \frac{p}{1-p}\,\mathsf{d}_{\mathsf{lin}}(M,M')\,. \tag{7}$$

This seems easy to prove in the case $\mathbb{P}_{\langle M',\pi\rangle}(\langle M',\pi\rangle\downarrow\underline{0}) = 0$: it suffices to observe that a path which is a successful reduction of $\langle (C^{\langle p\rangle})M,\varepsilon\rangle$ in the "Krivine Machine" of Section 2.1.2 (considered here as a Markov chain) can be decomposed as

$$\begin{aligned}\langle (C^{\langle p\rangle})M,\varepsilon\rangle &\to^* \langle \mathsf{if}(\mathsf{coin}(p),M,\Omega^\sigma),\pi_1(C,M)\rangle \to^* \langle \mathsf{if}(\mathsf{coin}(p),M,\Omega^\sigma),\pi_2(C,M)\rangle\\ &\to^* \cdots \to^* \langle \mathsf{if}(\mathsf{coin}(p),M,\Omega^\sigma),\pi_k(C,M)\rangle \to^* \langle\underline{0},\varepsilon\rangle\end{aligned}$$

where $(\pi_i(C,M))_{i=1}^k$ is a finite sequence of stacks such that $\sigma\vdash\pi_i(M)$ for each i. Notice that this sequence of stacks depends not only on C and M but also on the considered path of the Markov chain.

In the general case, Inequation (7) seems less easy to prove because, for a given common initial context C, the sequences of reductions (and of associated stacks) starting with $\langle (C^{\langle p\rangle})M,\varepsilon\rangle$ and $\langle (C^{\langle p\rangle})M',\varepsilon\rangle$ differ. This divergence has low probability when $\mathsf{d}_{\mathsf{lin}}(M,M')$ is small, but it is not completely clear how to evaluate it. Coinductive methods like probabilistic bisimulation as in the work of Crubillé and Dal Lago are certainly relevant here.

Our Theorem 10 shows that another and more geometric approach, based on a simple denotational model, is also possible to get Theorem 11 which, though weaker than Inequation (7), allows nevertheless to control the p-tamed distance.

We finish the paper by observing that the equivalence relations induced on terms by these observational distances coincide with the ordinary observational distance if $p\neq 0$.

▶ **Theorem 12.** *Assume that $0<p\leq 1$. If $\mathsf{d}_{\mathsf{obs}}^{\langle p\rangle}(M,M')=0$ then $M\sim M'$ (that is, M and M' are observationally equivalent).*

Proof. If $\vdash M:\sigma$ we set $M_p = \mathsf{if}(\mathsf{coin}(p),M,\Omega^\sigma)$. If $\mathsf{d}_{\mathsf{obs}}^{\langle p\rangle}(M,M')=0$ then $M_p\sim M'_p$ by definition of observational equivalence, hence $[\![M_p]\!]=[\![M'_p]\!]$ by our Full Abstraction Theorem [10], but $[\![M_p]\!]=p[\![M]\!]$ and similarly for M'. Since $p\neq 0$ we get $[\![M]\!]=[\![M']\!]$ and hence $M\sim M'$ by adequacy [10]. ◀

So for each $p \in (0,1)$ and for each type σ we can consider $\mathsf{d}^{\langle p \rangle}$ as a distance on the observational classes of closed terms of type σ. We call it the *p-tamed observational distance*. Our Theorem 11 shows that we can control this distance using the denotational distance. For instance we have $\mathsf{d}_{\mathsf{obs}}^{\langle p \rangle}(\mathsf{coin}(0), \mathsf{coin}(\varepsilon)) \leq \frac{2p\varepsilon}{1-p}$ so that $\mathsf{d}_{\mathsf{obs}}^{\langle p \rangle}(\mathsf{coin}(0), \mathsf{coin}(\varepsilon))$ tends to 0 when ε tends to 0.

Conclusion

The two results of this paper are related: both use derivatives wrt. probabilities to evaluate the number of times arguments are used. We think that they provide motivations for investigating further differential extensions of pPCF and related languages in the spirit of [11].

References

1. Johannes Borgström, Ugo Dal Lago, Andrew D. Gordon, and Marcin Szymczak. A lambda-calculus foundation for universal probabilistic programming. In Jacques Garrigue, Gabriele Keller, and Eijiro Sumii, editors, *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, ICFP 2016, Nara, Japan, September 18-22, 2016*, pages 33–46. ACM, 2016.
2. Simon Castellan, Pierre Clairambault, Hugo Paquet, and Glynn Winskel. The concurrent game semantics of Probabilistic PCF. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 215–224. ACM, 2018. doi:10.1145/3209108.
3. Raphaëlle Crubillé. Probabilistic Stable Functions on Discrete Cones are Power Series. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 275–284. ACM, 2018. doi:10.1145/3209108.3209198.
4. Raphaëlle Crubillé and Ugo Dal Lago. Metric Reasoning About Lambda-Terms: The General Case. In Hongseok Yang, editor, *Programming Languages and Systems - 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10201 of *Lecture Notes in Computer Science*, pages 341–367. Springer, 2017. doi:10.1007/978-3-662-54434-1_13.
5. Vincent Danos and Thomas Ehrhard. Probabilistic coherence spaces as a model of higher-order probabilistic computation. *Information and Computation*, 152(1):111–137, 2011.
6. Vincent Danos and Russell Harmer. Probabilistic game semantics. In *Proceedings of the 15th Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society, 2000.
7. Daniel de Carvalho. Execution Time of lambda-Terms via Denotational Semantics and Intersection Types. *CoRR*, abs/0905.4251, 2009. arXiv:0905.4251.
8. Daniel de Carvalho. Execution time of λ-terms via denotational semantics and intersection types. *MSCS*, 28(7):1169–1203, 2018.
9. Thomas Ehrhard. An introduction to differential linear logic: proof-nets, models and antiderivatives. *Mathematical Structures in Computer Science*, 28(7):995–1060, 2018. doi:10.1017/S0960129516000372.
10. Thomas Ehrhard, Michele Pagani, and Christine Tasson. Full Abstraction for Probabilistic PCF. *Journal of the ACM*, 65(4):23:1–23:44, 2018. doi:10.1145/3164540.
11. Thomas Ehrhard and Laurent Regnier. The differential lambda-calculus. *Theoretical Computer Science*, 309(1-3):1–41, 2003.
12. Jean-Yves Girard. Between logic and quantic: a tract. In Thomas Ehrhard, Jean-Yves Girard, Paul Ruet, and Philip Scott, editors, *Linear Logic in Computer Science*, volume 316 of *London Mathematical Society Lecture Notes Series*, pages 346–381. Cambridge University Press, 2004.

13 Klaus Keimel and Gordon D. Plotkin. Mixed powerdomains for probability and nondeterminism. *Logical Methods in Computer Science*, 13(1), 2017. doi:10.23638/LMCS-13(1:2)2017.

14 Peter Selinger. Towards a semantics for higher-order quantum computation. In *Proceedings of the 2nd International Workshop on Quantum Programming Languages, Turku, Finland*, number 33 in TUCS General Publication. Turku Centre for Computer Science, 2004.

15 Matthijs Vákár, Ohad Kammar, and Sam Staton. A domain theory for statistical probabilistic programming. *PACMPL*, 3(POPL):36:1–36:29, 2019.

Modal Embeddings and Calling Paradigms

José Espírito Santo
Centre of Mathematics, University of Minho, Portugal
jes@math.uminho.pt

Luís Pinto
Centre of Mathematics, University of Minho, Portugal
luis@math.uminho.pt

Tarmo Uustalu
School of Computer Science, Reykjavik University, Iceland
Dept. of Software Science, Tallinn University of Technology, Estonia
tarmo@ru.is

Abstract

We study the computational interpretation of the two standard modal embeddings, usually named after Girard and Gödel, of intuitionistic logic into IS4. As source system we take either the call-by-name (cbn) or the call-by-value (cbv) lambda-calculus with simple types. The target system can be taken to be the, arguably, simplest fragment of IS4, here recast as a very simple lambda-calculus equipped with an indeterminate lax monoidal comonad. A slight refinement of the target and of the embeddings shows that: the target is a calculus indifferent to the calling paradigms cbn/cbv, obeying a new paradigm that we baptize call-by-box (cbb), and enjoying standardization; and that Girard's (resp. Gödel's) embbedding is a translation of cbn (resp. cbv) lambda-calculus into this calculus, using a compilation technique we call protecting-by-a-box, enjoying the preservation and reflection properties known for cps translations - but in a stronger form that allows the extraction of standardization for cbn or cbv as consequence of standardization for cbb. The modal target and embeddings achieve thus an unification of call-by-name and call-by-value as call-by-box.

2012 ACM Subject Classification Theory of computation → Logic; Theory of computation → Program semantics

Keywords and phrases intuitionistic S4, call-by-name, call-by-value, comonadic lambda-calculus, standardization, indifference property

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.18

Funding J.E.S. and L.P. were supported by Fundação para a Ciência e a Tecnologia (FCT) through project UID/MAT/00013/2013. T.U. was supported by the Estonian Ministry of Education and Research through institutional research grant IUT33-13. All three authors received support from the COST action CA15123 EUTYPES.

1 Introduction

It is a fact reported in textbooks [16] that there are two main embeddings of intuitionistic logic into (intuitionistic) modal logic S4, the original one due to Gödel and a more recent one named after Girard. What is the computational meaning of this fact? In particular, why two? Similar questions concerning the embedding of intuitionistic logic into linear logic have been answered long ago: the $(!A \multimap B)$- and $!(A \multimap B)$-translations already introduced in the seminal paper [4] correspond to the two calling mechanisms of functional programming, call-by-name (cbn) and call-by-value (cbv), which are thus "explained in terms of logical translations, bringing them into the scope of the Curry-Howard isomorphism" [11]. Through these results in terms of linear logic we can glimpse what the computational explanation of the embeddings into modal logic is.

© José Espírito Santo, Luís Pinto, and Tarmo Uustalu;
licensed under Creative Commons License CC-BY
4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019).
Editor: Herman Geuvers; Article No. 18; pp. 18:1–18:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The claim of the present paper is that it is desirable to give a direct analysis of the embeddings into modal logic S4. First, such analysis is more abstract, because it is done in terms of an indeterminate $\Box$-modality, and so the analysis applies to all the possible instantiations of the modality, including the ! modality of linear logic. Second, such analysis will be carried farther than what was done before with linear logic. Here is what we obtained:

First, the target of the modal embeddings can be defined, in a first moment, as the simplest fragment of intuitionistic S4 where the problem of closure under substitution is solved. This target can be presented as a very simple λ-calculus equipped with an indeterminate lax monoidal comonad. In a second moment, the target should be slightly refined into a λ-calculus with several noteworthy properties: (i) it follows a new calling mechanism, *call-by-box*; (ii) it is equipped with a notion of evaluation, *weak-and-external* reduction, which is *indifferent* to cbn and cbv; (iii) it enjoys a standardization theorem that makes explicit the contribution of evaluation to a notion of standard reduction. In the description of Plotkin [12], the target system is a calculus and a programming language, and the standardization theorem links both. We say all these ingredients turn call-by-box (cbb) into a new *calling paradigm.*

Second, the embeddings can be defined, as expected, on the cbn λ-calculus (in the case of Girard) and on Plotkin's cbv λ-calculus (in the case of Gödel). But, after a refinement of Gödel's embedding, both can be seen as having as target the above refined target. When this is done, the embeddings can be described as a compilation of cbn or cbv into cbb, following a new technique that we call *protecting-by-a-box.* This technique improves the old protecting-by-a-lambda, already discussed in [12], which only achieves the compilation of cbn into cbv. *Au contraire*, protecting-by-a-box is capable of compiling both cbn and cbv into the indifferent paradigm cbb. In addition, the refined embeddings enjoy properties of preservation and reflection at the levels of reduction and evaluation, and also standard reduction. So all the *translation, simulation, and indifference properties* of cps-translations [12] hold of the refined embeddings, and so they can be offered as an improvement of protecting-by-a-lambda alternative to cps-translations.

Third, the indifference property of the cbb target, which at first only means that reduction or evaluation in the full target captures cbn (resp. cbv) reduction or evaluation when restricted to the image of Girard's (resp. Gödel's) embedding, actually goes much further: all the translation, simulation, and indifference properties cooperate to show that the standardization theorems for the cbn and cbv λ-calculi can be extracted from the standardization theorem of the cbb target. Because of all this, we feel entitled to say that the cbb target, together with the refined modal embeddings, achieve a *modal unification of call-by-name and call-by-value*

Plan of the paper. Section 2 recalls the cbn (i.e. ordinary) λ-calculus $\lambda_{\mathtt{n}}$ and Plotkin's cbv λ-calculus $\lambda_{\mathtt{v}}$. Section 3 recasts the modal embeddings as maps from $\lambda_{\mathtt{n}}$ or $\lambda_{\mathtt{v}}$ into a simple, modal target language $\lambda_{\Box}$. Section 4 motivates and introduces the refined target $\lambda_{\mathtt{b}}$ and proves that it enjoys standardization. Section 5 introduces the refined embeddings and proves their properties. Section 6 briefly shows how to instantiate our results with two $\Box$-modalities. Section 7 concludes.

2 Background

The source calculi of the modal translations we will study in this paper are the call-by-name and call-by value λ-calculi. In this section we fix notation, terminology and several definitions regarding these calculi, including what we mean by a "calling paradigm", and by "indifference property", and how we define standard reduction.

$$\frac{}{\Gamma, x : A \vdash x : A} \qquad \frac{\Gamma, x : A_1 \vdash M : A_2}{\Gamma \vdash \lambda x.M : A_1 \supset A_2} \qquad \frac{\Gamma \vdash M : A_1 \supset A_2 \quad \Gamma \vdash N : A_1}{\Gamma \vdash MN : A_2}$$

Figure 1 (Shared) typing rules of source calculi $\lambda_{\mathtt{n}}$ and $\lambda_{\mathtt{v}}$.

The modal translations will be defined on untyped source calculi, but at the same time we will develop simply-typed versions of the source calculi and translations. The source calculi are based on the set of λ-terms, given by

$$M, N, P, Q ::= x \mid \lambda x.M \mid MN$$

A value is a term of the form x or $\lambda x.M$. Values are ranged over by V, W.

We consider two reduction rules

$$(\lambda x.M)N \to [N/x]M \qquad (\beta_{\mathtt{n}}) \qquad\qquad (\lambda x.M)V \to [V/x]M \qquad (\beta_{\mathtt{v}})$$

As usual, $\to_{\beta_{\mathtt{n}}}$ (resp. $\to_{\beta_{\mathtt{v}}}$) denotes the compatible closure of $\beta_{\mathtt{n}}$ (resp. $\beta_{\mathtt{v}}$). Compatible closure is the closure under the term formers for λ-abstraction and application, *i.e.* closure under the rules:

$$\frac{M \to M'}{MN \to M'N}\ (\mu) \qquad \frac{N \to N'}{MN \to MN'}\ (\nu) \qquad \frac{M \to M'}{\lambda x.M \to \lambda x.M'}\ (\xi)$$

When we equip the λ-terms with $\to_{\beta_{\mathtt{n}}}$ we obtain the ordinary λ-calculus, or **call-by-name λ-calculus** here denoted $\lambda_{\mathtt{n}}$; when we equip the λ-terms with $\to_{\beta_{\mathtt{v}}}$ we obtain Plotkin's **cbv λ-calculus** [12], here denoted $\lambda_{\mathtt{v}}$.

We will develop in parallel the typed version of these calculi. Here, types are given by:

$$A, A' ::= X \mid A \supset A'$$

Let Γ range over sets of type assignments $x : A$ with all x distinct. The typing system derives sequents of the form $\Gamma \vdash M : A$. The typing rules are given in Fig. 1. Logically, this is a presentation of intuitionistic implicational logic.

We define sub-relations of $\to_{\beta_{\mathtt{n}}}$ and $\to_{\beta_{\mathtt{v}}}$. To this end, we need the closure rule $\nu_<$, the restriction of ν above where M is V. Then we define: $\to_{\mathtt{w}}$ as $\beta_{\mathtt{n}}$ closed under μ and ν; $\to_{\mathtt{n}}$ as $\beta_{\mathtt{n}}$ closed under μ; $\to_{\mathtt{v}}$ as $\beta_{\mathtt{v}}$ closed under μ and $\nu_<$.

In $\to_{\mathtt{w}}$, reduction under λ's is forbidden, it is in this sense that $\to_{\mathtt{w}}^*$ is called **weak reduction**. In $\to_{\mathtt{w}}$, reduction in applications can occur both in function position or argument position, in any order. Relations $\to_{\mathtt{n}}$ and $\to_{\mathtt{v}}$ are two ways of restricting $\to_{\mathtt{w}}$ to get a deterministic relation (a partial function). The effect of $\nu_<$ is to force reduction in function position first (as reduction in arguments can only occur when the function position term is already a value). This option we call *left-first* and convey the idea with the symbol $<$. Notice $\nu_<$ has to be combined with $\beta_{\mathtt{v}}$: closing $\beta_{\mathtt{n}}$ under μ and $\nu_<$ does not give a deterministic relation (since a $\beta_{\mathtt{n}}$ redex with a non-value argument can reduce in two ways in this relation). We call $\to_{\mathtt{n}}^*$ and $\to_{\mathtt{v}}^*$ respectively **call-by-name evaluation** and **call-by-value evaluation**. Weak reduction and cbn evaluation make sense in $\lambda_{\mathtt{n}}$ while cbv evaluation makes sense in $\lambda_{\mathtt{v}}$.

The **standardization** theorem for $\lambda_{\mathtt{n}}$ says that $M \to_{\beta_{\mathtt{n}}}^* N$ iff M reduces in a *standard* way to N; it states the completeness of that standard way of reducing. The specification of the standard way of reducing can be made by characterizing what reduction sequences are accepted as standard [1], or by axiomatizing the relation that M reduces in a standard

$$\frac{}{x \Rrightarrow_{\mathtt{n}} x}\ VAR \qquad \frac{M \Rrightarrow_{\mathtt{n}} N}{\lambda x.M \Rrightarrow_{\mathtt{n}} \lambda x.N}\ ABS \qquad \frac{M \Rrightarrow_{\mathtt{n}} M' \quad N \Rrightarrow_{\mathtt{n}} N'}{MN \Rrightarrow_{\mathtt{n}} M'N'}\ APL$$

$$\frac{M \to_{\mathtt{n}}^{*} \lambda x.M' \quad [N/x]M' \Rrightarrow_{\mathtt{n}} P}{MN \Rrightarrow_{\mathtt{n}} P}\ RDX$$

Figure 2 Standard reduction in $\lambda_{\mathtt{n}}$.

$$\frac{}{x \Rrightarrow_{\mathtt{v}} x}\ VAR \qquad \frac{M \Rrightarrow_{\mathtt{v}} N}{\lambda x.M \Rrightarrow_{\mathtt{v}} \lambda x.N}\ ABS \qquad \frac{M \Rrightarrow_{\mathtt{v}} M' \quad N \Rrightarrow_{\mathtt{v}} N'}{MN \Rrightarrow_{\mathtt{v}} M'N'}\ APL$$

$$\frac{M \to_{\mathtt{v}}^{*} \lambda x.M' \quad N \to_{\mathtt{v}}^{*} V \quad [V/x]M' \Rrightarrow_{\mathtt{v}} P}{MN \Rrightarrow_{\mathtt{v}} P}\ RDX$$

Figure 3 Standard reduction in $\lambda_{\mathtt{v}}$.

way to N [8]. In Fig. 2 we give one such axiomatization (it thus is in the spirit of [8], but notice no use is made of the vector notation), with the relation denoted $M \Rrightarrow_{\mathtt{n}} N$. It is straightforward to see that $\Rrightarrow_{\mathtt{n}}$ is contained in $\to_{\beta_{\mathtt{n}}}^{*}$, and from such a proof one extracts a notion of **standard reduction sequence**: it starts with cbn evaluation (corresponding to applications of rule RDX) of the given application MN, until one decides to freeze the outer construct and do reduction inside the subexpressions (corresponding to application of the other rules). The inclusion of $\to_{\beta_{\mathtt{n}}}^{*}$ in $\Rrightarrow_{\mathtt{n}}$ is the real content of the standardization theorem, and will be obtained later as a particular case of a more general, unifying result.

We also give a definition of the relation $M \Rrightarrow_{\mathtt{v}} N$ (M reduces in a standard way to N in $\lambda_{\mathtt{v}}$), again not by characterizing standard reduction sequences [12], rather by the inductive definition in Fig. 3. These rules determine a similar notion of standard reduction sequence: cbv evaluation of the given application MN, until one decides to freeze the outer construct and do reduction inside the subexpressions. The standardization theorem for $\lambda_{\mathtt{v}}$ will also be obtained later as a particular case of the same more general, unifying result.

Call-by-name and call-by-value are **calling paradigms**, in the sense that each comprises: a variant of the β-rule, specifying how functions are called, and generating a notion of deterministic evaluation and a notion of full reduction. According to [12], the evaluation relation can be understood as a programming language, and the full reduction (more precisely, the related notion of equality) can be understood as the corresponding calculus. The standardization theorem shows how evaluation can be used in a specific, but complete, way of reducing, and thereby links the programming language and the calculus.

Call-by-name and call-by-value are related by cps-translations, one from cbn to cbv, another the other way around [12]. The maps link full reduction or evaluation in the source system to full reduction or evaluation in the target system, respectively, and these are the **translation** and **simulation** properties of the maps [12]. But the target of the cps-translations is the subset of λ-terms given by the grammar

$$M, N ::= V \mid MV \qquad\qquad V ::= x \mid \lambda x.M$$

This is such a restricted set of terms that we cannot observe any difference between $\beta_{\mathtt{n}}$- and $\beta_{\mathtt{v}}$-reduction, and that all three relations $\to_{\mathtt{w}}$, $\to_{\mathtt{n}}$, and $\to_{\mathtt{v}}$ collapse to the same relation, viz. $\beta_{\mathtt{v}}$ closed under μ – a kind of intersection between $\to_{\mathtt{n}}$ and $\to_{\mathtt{v}}$. So, in this subset of terms

$$\frac{}{\Gamma, x : B \vdash x : B} \qquad \frac{\Gamma, x : B \vdash M : A}{\Gamma \vdash \lambda x.M : B \supset A} \qquad \frac{\Gamma \vdash M : B \supset A \quad \Gamma \vdash N : B}{\Gamma \vdash MN : A}$$

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \mathsf{box}(M) : \Box A} \qquad \frac{\Gamma \vdash M : \Box A}{\Gamma \vdash \varepsilon(M) : A}$$

Figure 4 Typing rules of the modal target calculus $\lambda_\Box$.

we cannot observe any difference between cbn and cbv evaluation; and weak reduction is also deterministic and coincides with those. This is the **indifference property** of the image of cps-translations.

Modal embeddings, we will see, also provide translations and simulations of cbn and cbv into a shared target enjoying an indifference property and following a new calling paradigm.

3 Modal embeddings

In this section, we recast the two modal embeddings of intuitionistic logic into intuitionistic modal logic S4 due to Girard and Gödel [16] as translations from $\lambda_{\mathtt{n}}$ and $\lambda_{\mathtt{v}}$ into a very simple λ-calculus, named $\lambda_\Box$. This system corresponds to a fragment of intuitionistic S4, but will prove to be strong enough to interpret call-by-name and call-by-value. We will recall the well-known properties of preservation of reduction by the embeddings in considerable detail, since this will be useful to motivate the refinements in the following sections.

3.1 Modal target calculus $\lambda_\Box$

We will develop in parallel the untyped and typed versions of the comonadic language $\lambda_\Box$. The terms are given by:

$$M, N, P, Q ::= x \mid \lambda x.M \mid MN \mid \mathsf{box}(M) \mid \varepsilon(M)$$

On this set we define two reduction rules:

$$(\lambda x.M)N \to [N/x]M \qquad (\beta_\supset) \qquad\qquad \varepsilon(\mathsf{box}(M)) \to M \qquad (\beta_\Box)$$

Here $[N/x]M$ denotes ordinary substitution.

As usual, for $R \in \{\beta_\supset, \beta_\Box\}$ or $R = \beta_\supset \cup \beta_\Box$, $\to_R$ denotes the closure of R under all term formers. In other words, $\to_R$ denotes the closure of R under five rules: rules μ, ν and ξ, allowing reduction under application or abstraction, plus two rules allowing reduction under box and ε. Instead of $\to_{\beta_\supset \cup \beta_\Box}$ normally we only write $\to$; so, in this case, $\to^*$ (resp. $\to^+$) stands for the reflexive-transitive (resp. transitive) closure of $\to_{\beta_\supset \cup \beta_\Box}$.

Typing helps grasping this term language. Types are given by:

$$A ::= X \mid B \supset A \mid B \qquad\qquad B ::= \Box A$$

Contexts Γ are sets of declarations $x : B$ where each x is declared at most once. The typing system derives sequents $\Gamma \vdash M : A$, and the typing rules are in Fig. 4.

The simple-minded $\Box$ introduction rule is appropriate because of the restriction of contexts to boxed types (for arbitrary contexts this rule is unsound for intuitionistic S4). This restriction, in turn, dictates the restriction of left-hand-sides of implications to boxed types.

$$\begin{array}{rcl} x^{\sharp}_U & = & x \\ (\lambda x.\, M)^{\sharp}_U & = & \lambda x. M^{\sharp}_{U,x} \\ (MN)^{\sharp}_U & = & M^{\sharp}_U N^{\sharp}_U \end{array} \qquad \begin{array}{rcl} (\mathsf{box}(M))^{\sharp}_U & = & \mathsf{cobind}(x_1,\dots,x_n, x_1 \dots x_n . M^{\sharp}_U) \\ & & (x_1,\dots,x_n = U, \text{ all } x_i \text{ distinct}) \\ (\varepsilon(M))^{\sharp}_U & = & \varepsilon(M^{\sharp}_U) \end{array}$$

Figure 5 Scoped term translation from $\lambda_\Box$ to term calculus of **IS4**.

The $\beta_\Box$ reduction rule corresponds to the rule for contracting $\Box$ introduction/elimination detours.

Substitution enjoys the usual admissible typing rule, but with a restriction to boxed types imposed by the definition of contexts:

$$\frac{\Gamma \vdash N : B \quad \Gamma, x : B \vdash M : A}{\Gamma \vdash [N/x]M : A} \tag{1}$$

The proof uses admissibility of weakening in the case $M = \lambda y.M'$. The latter follows by an immediate induction, but crucially depends on the restriction of contexts to boxed types.

▶ **Proposition 1** (Subject reduction of $\lambda_\Box$). *In $\lambda_\Box$, if $M \to N$ and $\Gamma \vdash M : A$, then $\Gamma \vdash N : A$.*

Proof. By induction on $\to$. The base case relative to $\beta_\supset$ uses the typing rule for substitution (1). The base case relative to $\beta_\Box$ is immediate by inversion of the typing rules for ε and box. ◀

3.2 Comparison of $\lambda_\Box$ with IS4

The difficulty of formulating a satisfactory natural deduction system for the modal logic S4 is well known since Prawitz [13]. The issue is not at the level of provability, but rather at the level of normalization: to guarantee that the system is closed under substitution [2]. Bierman and de Paiva [2], in their natural deduction system **IS4** for full intuitionistic S4 (recalled in Appendix A), need to work with the general introduction rule for the modality, which brings certain complications. In contrast, in $\lambda_\Box$, we adopt the naive introduction rule for the modality, but have to operate with the restricted implications and sequents that ensure closure under substitution.

Therefore, on the level of logic alone, $\lambda_\Box$ is a fragment of Bierman and de Paiva's **IS4**. As a term calculus, $\lambda_\Box$ is a fragment of Bierman and de Paiva's term calculus, which is a calculus for a cartesian closed category equipped with a lax monoidal comonad. Fig. 5 gives a translation of scoped $\lambda_\Box$ terms to scoped **IS4** terms. U ranges over sets of variables; $M^{\sharp}_U$ is well-defined if $\mathrm{FV}(M) \subseteq U$. We write $|\Gamma|$ for the set $\{x \mid x : A \in \Gamma\}$. This translation not only preserves typing and reduction steps of $\lambda_\Box$ terms, but also *reflects* typing and reduction steps of the **IS4** terms in its image, thus isolating a fragment of **IS4** isomorphic to $\lambda_\Box$:

▶ **Proposition 2** (Preservation and reflection of typing and reduction from $\lambda_\Box$ to IS4).

1. *For all Γ, A in $\lambda_\Box$, $\Gamma \vdash M : A$ in $\lambda_\Box$ iff $\Gamma \vdash M^{\sharp}_{|\Gamma|} : A$ in the term calculus for* **IS4**.
2. *For all U s. t. $\mathrm{FV}(M) \subseteq U$, $M \to N$ in $\lambda_\Box$ iff $M^{\sharp}_U \to N^{\sharp}_U$ in the term calculus for* **IS4**.

▶ **Proposition 3** (Conservativity of IS4 over $\lambda_\Box$). *For all Γ, A in $\lambda_\Box$, if $\Gamma \vdash M : A$ in the term calculus for* **IS4**, *then there exists N such that $\Gamma \vdash N : A$ in $\lambda_\Box$.*

See the appendix for proofs.[1]

[1] In the case $\mathsf{box}(M)$ of the translation in Fig. 5, it is important to "rebind" all variables of M to

$$\begin{array}{rcl} X^\circ & = & X \\ (A_1 \supset A_2)^\circ & = & \Box A_1^\circ \supset A_2^\circ \end{array} \qquad \begin{array}{rcl} x^\circ & = & \varepsilon(x) \\ (\lambda x.M)^\circ & = & \lambda x.M^\circ \\ (MN)^\circ & = & M^\circ \mathsf{box}(N^\circ) \end{array}$$

Figure 6 Translation from λ_{n} to $\lambda_\Box$ ("Girard's translation").

3.3 Modal embeddings $(\cdot)^\circ$ and $(\cdot)^*$

The two modal translations are presented as maps from λ-terms to $\lambda_\Box$-terms. The original, and easier to grasp, motivation is logical, so the mapping of types and type preservation by the translations is presented right away. We also detail the preservation of reduction steps, which has been observed many times in many contexts [5, 11, 3]. Later, we will strengthen these results.

The translation from λ_{n} to $\lambda_\Box$ is in Fig. 6. On the level of types (i.e. the underlying logic) it is inspired by translation of intuitionistic logic into linear logic [4], based on the characteristic decomposition of intuitionistic implication into linear implication and the ! modality.[2]

In the following, $\Box\Gamma^\circ = x_1 : \Box A_1^\circ, \ldots, x_n : \Box A_n^\circ$ when $\Gamma = x_1 : A_1, \ldots, x_n : A_n$.

▶ **Proposition 4** (Preservation and reflection of typing by Girard's translation). *$\Gamma \vdash M : A$ in λ_{n} iff $\Box\Gamma^\circ \vdash M^\circ : A^\circ$ in $\lambda_\Box$.*

Proof. In each direction, routine induction on the given typing derivation. For example, in the "if" direction: the case $M = x$ follows by the axiom of λ_{n}, because the hypothesis implies $x : A$ is in Γ; the case $M = \lambda x.N$ follows because the hypothesis implies, for some A_1, A_2, $A = A_1 \supset A_2$ and $\Box\Gamma^\circ, x : \Box A_1^\circ \vdash N^\circ : A_2^\circ$ (through the immediate subderivation of the given typing derivation), so the IH applies and the $\supset$ introduction typing rule of λ_{n} can be used to conclude. ◀

▶ **Lemma 5.** $[\mathsf{box}(N^\circ)/x]M^\circ \to^*_{\beta_\Box} ([N/x]M)^\circ$.

Proof. By induction on M. The critical case $M = x$ reads: $LHS = \varepsilon(\mathsf{box}(N^\circ)) \to_{\beta_\Box} N^\circ = RHS$. ◀

▶ **Proposition 6** (Preservation of reduction by Girard's translation). *If $M \to_{\beta_{\mathsf{n}}} N$ in λ_{n}, then $M^\circ \to^+ N^\circ$ in $\lambda_\Box$.*

Proof. By induction on $M \to_{\beta_{\mathsf{n}}} N$. The base case uses the previous lemma:

$$((\lambda x.M)N)^\circ = (\lambda x.M^\circ)\mathsf{box}(N^\circ) \to_{\beta_\supset} [\mathsf{box}(N^\circ)/x]M^\circ \to^*_{\beta_\Box} ([N/x]M)^\circ \ . \tag{2}$$

◀

The translation from λ_{v} to $\lambda_\Box$ is in Fig. 7. A^* is denoted $A^\Box$ in [16], where it is defined directly by recursion on A: $X^\Box = \Box X$ and $(A_1 \supset A_2)^\Box = \Box(A_1^\Box \supset A_2^\Box)$. These are two styles for defining the same translation of types. The style we adopted is the same of Gödel's 1933 paper, while the alternative style was proposed by McKinsey-Tarski [15].

match the typing rule of cobind in **IS4** (introduction rule of $\Box$). Contrary to $\lambda_\Box$, contexts of **IS4** may contain unboxed formulas. The cobind typing rule is essentially a combination of the box typing rule (introduction rule of $\Box$ in $\lambda_\Box$), relying on a fully boxed context $x_1 : B_1, ..., x_n : B_n$, with a multicut.

[2] We could call Girard's translation the $(\Box A \supset B)$-translation.

$$
\begin{array}{rcl}
A^* & = & \Box A^\bullet \\
 & & \\
X^\bullet & = & X \\
(A_1 \supset A_2)^\bullet & = & \Box A_1^\bullet \supset \Box A_2^\bullet
\end{array}
\qquad
\begin{array}{rcl}
V^* & = & \mathsf{box}(V^\bullet) \\
(MN)^* & = & \varepsilon(M^*)N^* \\
x^\bullet & = & \varepsilon(x) \\
(\lambda x.M)^\bullet & = & \lambda x.M^*
\end{array}
$$

Figure 7 Translation from λ_{v} to $\lambda_\Box$ ("Gödel's translation").

At the term level, the translation is organized in two levels: there is a translation of terms M^* and a translation of values $V^\bullet$. A simpler translation with $x^* = x$ was possible, but the adopted version (with an η-expansion in the translation of variables) allows a uniform translation of values as terms: $V^* = \mathsf{box}(V^\bullet)$. It is sound to $\eta_\Box$-expand a variable of type A^* because A^* is a boxed type.[3]

In the following, $\Gamma^* = x_1 : A_1^*, \dots, x_n : A_n^*$ when $\Gamma = x_1 : A_1, \dots, x_n : A_n$.

▶ **Proposition 7** (Preservation and reflection of typing by Gödel's translation).

1. $\Gamma \vdash M : A$ *in* λ_{v} *iff* $\Gamma^* \vdash M^* : A^*$ *in* $\lambda_\Box$.
2. $\Gamma \vdash V : A$ *in* λ_{v} *iff* $\Gamma^* \vdash V^\bullet : A^\bullet$ *in* $\lambda_\Box$.

Proof. In each direction, the two statements are proved by mutual induction on the given typing derivation. ◀

▶ **Lemma 8.**

1. $[\mathsf{box}(V^\bullet)/x]M^* \to^*_{\beta_\Box} ([V/x]M)^*$.
2. $[\mathsf{box}(V^\bullet)/x]W^\bullet \to^*_{\beta_\Box} ([V/x]W)^\bullet$.

Proof. By simultaneous induction on M and W. The critical case $W = x$ reads: $LHS = \varepsilon(\mathsf{box}(V^\bullet)) \to_{\beta_\Box} V^\bullet = RHS$. ◀

▶ **Proposition 9** (Preservation of reduction by Gödel's translation). *If* $M \to_{\beta_{\mathsf{v}}} N$ *in* λ_{v}*, then* $M^* \to^+ N^*$ *in* $\lambda_\Box$.

Proof. By induction on $M \to_{\beta_{\mathsf{v}}} N$. The base case reads:

$$
\begin{array}{rcl}
((\lambda x.M)V)^* & = & \varepsilon(\mathsf{box}(\lambda x.M^*))\mathsf{box}(V^\bullet) \\
 & \to_{\beta_\Box} & (\lambda x.M^*)\mathsf{box}(V^\bullet) \to_{\beta_\supset} [\mathsf{box}(V^\bullet)/x]M^* \to^*_{\beta_\Box} ([V/x]M)^*
\end{array}
\tag{3}
$$

where the last reduction is justified by the previous lemma. ◀

▶ **Example 10.** Reflection of reduction along the translation from λ_{v} to $\lambda_\Box$ fails. Let $P := (\lambda x.xM)N$ (with $x \notin FV(M)$) and $Q := NM$. Then $P \to_{\beta_{\mathsf{v}}} Q$ does *not* hold in general. But $P^* \to^+ Q^*$ does hold, since:

$$
P^* = \varepsilon(\mathsf{box}(\lambda x.\varepsilon(\mathsf{box}(\varepsilon(x)))M^*))N^* \to^2_{\beta_\Box} (\lambda x.\varepsilon(x)M^*)N^* \to_{\beta_\supset} \varepsilon(N^*)M^* = Q^* \ .
$$

---

[3] We could call Gödel's translation the $(\Box A \supset \Box B)$-translation, and call McKinsey-Tarski translation the $(\Box(A \supset B))$-translation. There is again a connection with translations of intuitionistic logic into linear logic. The "call-by-value" translation of intuitionistic logic into linear logic is sometimes called the $!(A \multimap B)$-translation or the $(!A \multimap !B)$-translation. The former is found in the original paper by Girard [4]; the second is briefly mentioned by Lafont [7] in the discussion of the translation of λ-calculus, and used in [11]. In the translation of λ-terms into linear logic proofs and proof nets, Mackie [9] does the η-expansion of variables in the $(!A \multimap !B)$-translation and does not do it in the $!(A \multimap B)$-translation.

This example is adapted from one given in [14], where the same remark is made about the translation of λ_{v} into a linear λ-calculus (yet another, similar example is given in [11]). The path followed in [14] is to grow the source calculus from Plotkin's λ_{v} to Moggi's computational λ-calculus in order to derive more reductions. In the next section we follow a different path and shrink the target calculus.

4 Refined modal target calculus λ_{b}

In order to give a deeper analysis of the modal embeddings, and refine the results of the previous section, we will identify in this section a sublanguage λ_{b} of $\lambda_{\Box}$. In the next section, we give refined versions of the embeddings whose images lie in λ_{b}. In this section, we start with a motivation for the refinements. Next we define λ_{b} and prove some properties, notably a standardization theorem.

4.1 Motivation

We start by analyzing whether Proposition 6 and 9 could be stated as equivalences, so that we would also have reflection of reduction. By inspection of calculations (2) and (3), one recognizes an obstacle in the uncontrolled proliferation of $\beta_{\Box}$-reduction steps in the reduction between images. We regard $\beta_{\Box}$-reduction steps as **administrative** [12], and seek to hide them somehow. In the cps-literature, administrative steps are hidden by performing them at compile time by an optimized version of the translation [12]. Here, we will also use such an idea, but combined with another one: a refined definition of the target system will also avoid many administrative steps.

Inspecting (2) again one sees that administrative steps in the image of Girard's map come from Lemma 5. But notice that a substitution is always triggered with a box (a term of the form $\mathsf{box}(N)$) as the actual parameter; since in the target of Girard's map variables are always wrapped with ε, every actual replacement generates an administrative redex. A solution is to pass, not the box, but the contents of the box (the box is open), which will replace, not x, but $\varepsilon(x)$. The adoption of this special β-rule is a simple trick that eliminates all the administrative steps in the image of Girard's map.

How about Gödel's map? Inspecting calculation (3) one sees again that the $\beta_{\supset}$-redex has a box as argument, and that the subsequent administrative steps are justified by the lemma that shows how substitution commutes with the translation, namely Lemma 8. Again, a special β-reduction step could open the box before executing the substitution. But the $\beta_{\supset}$-reduction step has a preliminary administrative step, and not all occurrences of $\varepsilon(M)$ in the image of the map have the form $\varepsilon(x)$. The latter two problems have a common solution. In the translation of application in Fig. 7, one should put $(\lambda z.\varepsilon(z)N^*)M^*$. The preliminary administrative step in (3) will become a special β-reduction step, and $\varepsilon(x)$ will suffice in the images of the translation as a monolithic term form instead of x and $\varepsilon(M)$.

4.2 Definition of λ_{b} and relationship with $\lambda_{\Box}$

We call our refined modal calculus λ_{b}. Its terms are given by the grammar:

$$M, N, P, Q, T ::= \varepsilon(x) \mid \lambda x.M \mid MN \mid \mathsf{box}(N)$$

Note that variables x and ε are amalgamated, and constrained to the construction $\varepsilon(x)$. **Values** V are terms of the form $\varepsilon(x)$ or $\lambda x.M$. **Boxes** are terms of the form $\mathsf{box}(N)$, ranged over by B.

Types and sequents of $\lambda_{\mathtt{b}}$ are as for $\lambda_{\Box}$. Recall that in implications the antecedent must be a boxed type, and accordingly types in contexts must be boxed. The typing rules of $\lambda_{\mathtt{b}}$ are as for $\lambda_{\Box}$, except that the typing rule for $\varepsilon(x)$ corresponds to the obvious combination of the typing rules of $\lambda_{\Box}$ for variables and for ε, that reads as follows:

$$\frac{}{\Gamma, x : \Box A \vdash \varepsilon(x) : A}$$

Immediately: for any $M \in \lambda_{\mathtt{b}}$, $\Gamma \vdash M : A$ in $\lambda_{\mathtt{b}}$ iff $\Gamma \vdash M : A$ in $\lambda_{\Box}$.

The unique reduction rule of $\lambda_{\mathtt{b}}$ is:

$$(\lambda x.M)\mathsf{box}(N) \to [N/\varepsilon(x)]M \qquad (\beta_{\mathtt{b}})$$

where $[N/\varepsilon(x)]M$ is defined by recursion on M, and all clauses are homomorphic, except for the critical clauses:

$$[N/\varepsilon(x)]\varepsilon(x) = N \qquad\qquad [N/\varepsilon(x)]\varepsilon(y) = \varepsilon(y) \quad (x \neq y) \ .$$

As usual, $\to_{\beta_{\mathtt{b}}}$ denotes the compatible closure of $\beta_{\mathtt{b}}$, *i.e.* the closure of $\beta_{\mathtt{b}}$ under all term formers of $\lambda_{\mathtt{b}}$.

The $\beta_{\mathtt{b}}$-rule is a package of reduction steps of $\lambda_{\Box}$. In fact, for $M, N \in \lambda_{\mathtt{b}}$:

$$(\lambda x.M)\mathsf{box}(N) \to_{\beta_{\supset}} [\mathsf{box}(N)/x]M \to^*_{\beta_{\Box}} [N/\varepsilon(x)]M$$

where $[\mathsf{box}(N)/x]M \to^*_{\beta_{\Box}} [N/\varepsilon(x)]M$ is easily established by induction on M.

Rule $\beta_{\mathtt{b}}$ only fires when the argument is a box. For this reason we speak of **call-by-box**. In the typed setting, and since function types are always of the form $B \supset A$, arguments are always of boxed types – nevertheless, arguments are not necessarily boxes. We will prove call-by-box (abbreviated cbb) to be a calling paradigm, in the sense of Section 2, by defining evaluation and standard reduction and proving standardization.

We define sub-relations of $\to_{\beta_{\mathtt{b}}}$. To this end consider the closure rules:

$$\frac{M \to M'}{MN \to M'N}\ (\mu) \qquad \frac{M \to M'}{M\mathsf{B} \to M'\mathsf{B}}\ (\mu_{>}) \qquad \frac{N \to N'}{MN \to MN'}\ (\nu) \qquad \frac{N \to N'}{VN \to VN'}\ (\nu_{<})$$

Then: $\to_{\mathtt{we}}$ is inductively defined by $\beta_{\mathtt{b}}$ and μ and ν; $\to_{\mathtt{we}>}$ is inductively defined by $\beta_{\mathtt{b}}$ and $\mu_{>}$ and ν; $\to_{\mathtt{we}<}$ is inductively defined by $\beta_{\mathtt{b}}$, μ and $\nu_{<}$.

Notice: we always close the same β-rule (hence a single calling paradigm is at stake). Relation $\to_{\mathtt{we}}$ is called **weak** (because values do not reduce) and **external** - because boxes do not reduce. Relation $\to^*_{\mathtt{we}}$ is called **call-by-box evaluation**. Here, evaluation is taken in a relaxed sense, since the relation $\to_{\mathtt{we}}$ is non-deterministic: the cbb "evaluation" of a given application MN consists of the interleaved cbb "evaluation" of M and N in any order until a $\beta_{\mathtt{b}}$-redex emerges at root position. We may turn $\to_{\mathtt{we}}$ into a deterministic relation, by imposing either the left-first ($<$) or right-first ($>$) order of reduction in applications.[4] Cbn (resp. cbv) evaluation on $\lambda_{\mathtt{b}}$ will be defined later, as a restriction of $\to^*_{\mathtt{we}>}$ (resp. $\to^*_{\mathtt{we}<}$).

4.3 Properties of $\lambda_{\mathtt{b}}$

The main property of $\lambda_{\mathtt{b}}$ is how it unifies call-by-name and call-by-value, and will be seen in Section 5. Here we chose to show subject reduction, because it is a sanity check for a modal calculus, and standardization, because we want to promote call-by-box to a calling paradigm.

[4] Recall how the combination of rules $\beta_{\mathtt{n}}$, μ and $\nu_{<}$ on λ-terms failed to produce a deterministic relation.

$$\frac{}{\varepsilon(x) \Rightarrow_{\mathsf{b}} \varepsilon(x)}\ VAR \qquad \frac{M \Rightarrow_{\mathsf{b}} N}{\lambda x.M \Rightarrow_{\mathsf{b}} \lambda x.N}\ ABS \qquad \frac{M \Rightarrow_{\mathsf{b}} M' \quad N \Rightarrow_{\mathsf{b}} N'}{MN \Rightarrow_{\mathsf{b}} M'N'}\ APL \qquad \frac{M \Rightarrow_{\mathsf{b}} N}{\mathsf{box}(M) \Rightarrow_{\mathsf{b}} \mathsf{box}(N)}\ BOX$$

$$\frac{M \to^*_{\mathtt{we}} \lambda x.M' \quad N \to^*_{\mathtt{we}} \mathsf{box}(N') \quad [N'/\varepsilon(x)]M' \Rightarrow_{\mathsf{b}} P}{MN \Rightarrow_{\mathsf{b}} P}\ RDX$$

Figure 8 Standard reduction of λ_{b}.

$$\frac{}{M \Rightarrow_{\mathsf{b}} M}\ (1) \qquad \frac{}{(\lambda x.M)\mathsf{box}(N) \Rightarrow_{\mathsf{b}} [N/\varepsilon(x)]M}\ (2) \qquad \frac{M \Rightarrow_{\mathsf{b}} M' \quad N \Rightarrow_{\mathsf{b}} N'}{[N/\varepsilon(x)]M \Rightarrow_{\mathsf{b}} [N'/\varepsilon(x)]M'}\ (3)$$

$$\frac{M \to_{\mathtt{we}} N \Rightarrow_{\mathsf{b}} P}{M \Rightarrow_{\mathsf{b}} P}\ (4) \qquad \frac{M \to^*_{\mathtt{we}} N \Rightarrow_{\mathsf{b}} P}{M \Rightarrow_{\mathsf{b}} P}\ (5)$$

$$\frac{M \Rightarrow_{\mathsf{b}} \lambda x.M' \quad N \Rightarrow_{\mathsf{b}} \mathsf{box}(N')}{MN \Rightarrow_{\mathsf{b}} [N'/\varepsilon(x)]M'}\ (6) \qquad \frac{M \Rightarrow_{\mathsf{b}} (\lambda x.M')\mathsf{box}(N')}{M \Rightarrow_{\mathsf{b}} [N'/\varepsilon(x)]M'}\ (7) \qquad \frac{M \Rightarrow_{\mathsf{b}} N \to_{\beta_{\mathsf{b}}} P}{M \Rightarrow_{\mathsf{b}} P}\ (8)$$

Figure 9 Admissible rules of λ_{b}.

▶ **Proposition 11** (Subject reduction of λ_{b}). *In λ_{b}, if $M \to_{\beta_{\mathsf{b}}} N$ and $\Gamma \vdash M : A$, then $\Gamma \vdash N : A$.*

Proof. This is an immediate consequence of subject reduction for $\lambda_{\Box}$ (Prop. 1), and the facts (i) $\Gamma \vdash M : A$ in λ_{b} iff $\Gamma \vdash M : A$ in $\lambda_{\Box}$, and (ii) $M \to_{\beta_{\mathsf{b}}} N$ implies $M \to^*_{\beta} N$ in $\lambda_{\Box}$. ◀

Fig. 8 gives an inductive definition of the relation "M reduces in a standard way to N in λ_{b}", denoted $M \Rightarrow_{\mathsf{b}} N$.

▶ **Theorem 12** (Standardization of λ_{b}). *In λ_{b}, $M \to^*_{\beta_{\mathsf{b}}} N$ iff $M \Rightarrow_{\mathsf{b}} N$.*

Proof. Our proof is inspired in Loader's proof for λ-calculus [8], but notice no use of vector notation is made, and the definition of $\Rightarrow_{\mathsf{b}}$ makes explicit the contribution of evaluation.

The "if" direction is a very simple induction on $M \Rightarrow_{\mathsf{b}} N$ and just uses the facts that $\to^*_{\beta_{\mathsf{b}}}$ is reflexive, transitive and compatible, and that $\to^*_{\mathtt{we}} \subseteq \to^*_{\beta_{\mathsf{b}}}$.

The "only if" direction is proved by establishing the admissibility of the rules (1) to (8) in Fig. 9. Once this is done, the proof of the "only if" implication is by induction on $M \to^*_{\beta_{\mathsf{b}}} N$, and follows immediately from rules (1) and (8).

The proof of (1) is an easy induction on M. Then (2) follows from RDX and (1). The proof of (3) is by induction on $M \Rightarrow_{\mathsf{b}} M'$. The case RDX requires the substitution lemma for λ_{b}'s substitution, plus the following property of $\to_{\mathtt{we}}$: if $M \to_{\mathtt{we}} M'$ then $[N/\varepsilon(x)]M \to_{\mathtt{we}} [N/\varepsilon(x)]M'$. Rule (5) follows easily from (4), and the latter is proved by induction on $M \to_{\mathtt{we}} N$. Rule (6) is proved by induction on $M \Rightarrow_{\mathsf{b}} \lambda x.M'$ with subinduction on $N \Rightarrow_{\mathsf{b}} \mathsf{box}(N')$. Use is made of (3) and (5). Then, (7) follows easily from (6) by induction on $M \Rightarrow_{\mathsf{b}} (\lambda x.M')\mathsf{box}(N')$. Finally, (8) is proved by induction on $M \Rightarrow_{\mathsf{b}} N$, and uses (7). ◀

From the proof of the "if" implication of this theorem, one extracts a notion of **standard reduction sequence**: it starts with cbb evaluation (corresponding to applications of rule RDX) of the given application MN, until one decides to freeze the outer construct and do reduction inside the subexpressions (corresponding using the other rules in Fig. 8).

5 Refined embeddings into $\lambda_{\mathtt{b}}$

Continuing to implement the refinements motivated at the beginning of Section 4, we now introduce variants of Girard's and Gödel's translations from, respectively, $\lambda_{\mathtt{n}}$ and $\lambda_{\mathtt{v}}$ into the refined target calculus $\lambda_{\mathtt{b}}$. The hope was to improve Propositions 6 and 9, achieving reflection, in addition to preservation, of reduction by the maps.

In fact, we will obtain much more, as preservation and reflection will work at the levels of reduction, evaluation, and standard reduction – see Theorems 14 and 18 below. The results at the level of reduction and evaluation correspond to the **translation property** and **simulation property** of the maps, in the terminology of cps-translations [12]. The image of each map enjoys its own **indifference property**, and these two properties together are the indifference property of $\lambda_{\mathtt{b}}$. What is new compared to cps-translations is that the simulation and indifference properties cooperate to accomplish the results at the level of standard reduction; and the latter, together with the standardization for $\lambda_{\mathtt{b}}$, and the results at the level of reduction allow the extraction of the standardization for $\lambda_{\mathtt{n}}$ or $\lambda_{\mathtt{v}}$ as corollaries.

Translation of types stays unchanged both for the refined Girard's and Gödel's translations. At the level of terms, Girard's translation stays the same, but Gödel's translation will suffer a slight refinement, as promised at the beginning of Section 4. We reuse the symbols $(\cdot)^\circ$, $(\cdot)^*$ and $(\cdot)^\bullet$.

5.1 Refined Girard's embedding

We start with Girard's translation. Fig. 6 can be read *ipsis verbis* as defining a translation from $\lambda_{\mathtt{n}}$ to $\lambda_{\mathtt{b}}$. The refinement comes, not from the translation, but from the refined functioning of the target system. Preservation and reflection of typing, as stated in Prop. 4 for $\lambda_{\Box}$ for the original Girard's translation, holds in the same way for $\lambda_{\mathtt{b}}$ for the refined translation.

The image of the term translation is the subset of $\lambda_{\mathtt{b}}$ terms given by the grammar

$$M, N ::= \varepsilon(x) \mid \lambda x.M \mid M\mathsf{box}(N) \tag{4}$$

Let us call this subset **Girard's image**.

Due to the restricted form of arguments in applications, relations $\to_{\mathtt{we}}$ and $\to_{\mathtt{we}>}$ collapse on Girard's image to the same relation, one which can alternatively be defined as $\beta_{\mathtt{b}}$ closed under μ. This property of Girard's image we call its **indifference property**, by analogy with our account of the indifference property on λ-terms given at the end of Section 2. This single deterministic relation on Girard's image is denoted $\to_{\mathtt{n}}$. By **call-by-name evaluation** on $\lambda_{\mathtt{b}}$ we mean $\to_{\mathtt{n}}^*$. The terminology is justified by Theorem 14 below.

▶ **Lemma 13.** $[N^\circ/\varepsilon(x)]M^\circ = ([N/x]M)^\circ$.

▶ **Theorem 14** (Properties of refined Girard's translation)**.**

1. *(Preservation and reflection of reduction)* $M \to_{\beta_{\mathtt{n}}} N$ *in* $\lambda_{\mathtt{n}}$ *iff* $M^\circ \to_{\beta_{\mathtt{b}}} N^\circ$ *in* $\lambda_{\mathtt{b}}$.
2. *(Preservation and reflection of evaluation)* $M \to_{\mathtt{n}} N$ *in* $\lambda_{\mathtt{n}}$ *iff* $M^\circ \to_{\mathtt{n}} N^\circ$ *in* $\lambda_{\mathtt{b}}$.
3. *(Preservation and reflection of standard reduction)* $M \Rightarrow_{\mathtt{n}} N$ *in* $\lambda_{\mathtt{n}}$ *iff* $M^\circ \Rightarrow_{\mathtt{b}} N^\circ$ *in* $\lambda_{\mathtt{b}}$.

Proof.

Proof of 1. The "only if" half is proved by induction on $M \to_{\beta_{\mathtt{n}}} N$. The base case uses Lemma 13: $((\lambda x.M)N)^\circ = (\lambda x.M^\circ)\mathsf{box}(N^\circ) \to_{\beta_{\mathtt{b}}} [N^\circ/\varepsilon(x)]M^\circ = ([N/x]M)^\circ$. The "if" half follows from this fact: the image of $(\cdot)^\circ$ is closed under reduction and any reduction

between images is an image of a source reduction. Symbolically: if $M^\circ \to_{\beta_b} N'$, then there is a λ-term N such that $N^\circ = N'$ and $M \to_{\beta_n} N$. The proof is by induction on the λ-term M.

Proof of 2. By inspection of the proof of 1.

Proof of 3. The "only if" implication is proved by induction on $M \Rightarrow_n N$. The case RDX uses item 2 of this theorem. To prove the "if" direction, one proves something stronger: if $M^\circ \Rightarrow_b Q$ in λ_b then there is $N \in \lambda_n$ such that $N^\circ = Q$ and $M \Rightarrow_n N$ in λ_n. The proof is by induction on $M^\circ \Rightarrow_b Q$. The case RDX, besides Lemma 13, uses a strong form of the indifference property of Girard's image: $M^\circ \to_{we} Q$ iff $M^\circ \to_n Q$; and a strong form of item 2 of the present theorem: if $M^\circ \to_n Q$ then there is $N \in \lambda_n$ such that $N^\circ = Q$ and $M \to_n N$ in λ_n. ◀

▶ **Corollary 15** (Standardization of λ_n). *In λ_n, $M \to^*_{\beta_n} N$ iff $M \Rightarrow_n N$.*

Proof. Follows from Thm. 12 (standardization for λ_b), and parts 1 and 3 of Thm. 14. ◀

5.2 Refined Gödel's embedding

Now we turn to Gödel's translation. We will make use of the abbreviation

$$\mathsf{raise}(M) := \lambda z.\varepsilon(z)M$$

We immediately remark the following derived rules:

$$\frac{\Gamma \vdash M : B}{\Gamma \vdash \mathsf{raise}(M) : (\Box(B \supset B')) \supset B'} \qquad \frac{N \to^*_{we} \mathsf{box}(N') \quad M \to^*_{we} \mathsf{box}(\lambda x.M')}{\mathsf{raise}(N)M \to^*_{we} [N'/\varepsilon(x)]M'} \tag{5}$$

The latter is proved by the following reduction sequence:

$$\mathsf{raise}(N)M \to^*_{we} \mathsf{raise}(N)\mathsf{box}(\lambda x.M') \to_{we} (\lambda x.M')N \to^*_{we} (\lambda x.M')\mathsf{box}(N') \to_{we} [N'/\varepsilon(x)]M' \tag{6}$$

Gödel's term translation introduced before in Fig. 7 is refined thus:

$$V^* = \mathsf{box}(V^\bullet) \qquad (MN)^* = \mathsf{raise}(N^*)M^* \qquad x^\bullet = \varepsilon(x) \qquad (\lambda x.M)^\bullet = \lambda x.M^*$$

As said, the translation of types remains unchanged, and Proposition 7 about preservation and reflection of typing holds again, now for λ_b.

The image of the translation is contained in the subset of λ_b terms given by the grammar

$$M ::= \mathsf{box}(V) \mid VM \qquad V ::= \varepsilon(x) \mid \lambda x.M \tag{7}$$

For the exact image, the V in VM should be constrained to $\mathsf{raise}(M')$. But the subset (7) has the advantage of being closed under $\to_{\beta_b}$. Let us allow ourselves the abuse of calling (7) **Gödel's image**.[5]

Due to the restricted form of the term in function position in applications, relations $\to_{we}$ and $\to_{we<}$ collapse on Gödel's image to the same relation, one which can alternatively

[5] Gödel's image can be obtained in another way. Notice Girard's image is the set of λ_b-terms where the following is valid: a term is a box iff it is the argument term of an application. Likewise, we might want to characterize Gödel's image as the set of λ_b-terms where the following is valid: a term is a value iff it is the function term of an application. But the latter has to be complemented with the imposition that the contents of boxes are values (otherwise, closure under reduction would not be guaranteed).

be defined as β_{b} closed under ν. This property of Gödel's image we call its **indifference property**, again by analogy with our account of the indifference property on λ-terms given at the end of Section 2, but also in analogy with what happens in Girard's image. This single deterministic relation on Gödel's image is denoted $\to_{\mathsf{v}}$. By **call-by-value evaluation** on λ_{b} we mean $\to_{\mathsf{v}}^*$. The terminology is justified by Theorem 18 below.

▶ **Lemma 16.** $[V^\bullet/\varepsilon(x)]M^* = ([V/x]M)^*$ *and* $[V^\bullet/\varepsilon(x)]W^\bullet = ([V/x]W)^\bullet$*.*

The refined Gödel embedding improves Proposition 9: as we will see below in items 1 and 3 of Theorem 18, we get rid of the proliferation of administrative steps, and even obtain reflection for full evaluation (evaluation until a value is output). However, the following example shows that reflection of (standard) reduction still does not hold in general.

▶ **Example 17.** Let us return to Example 10. Recall $P = (\lambda x.xM)N$ (with $x \notin FV(M)$), $Q = NM$, and $P \to_{\beta_{\mathsf{v}}} Q$ does not hold in general. With refined Gödel's translation, it is still the case that $P^* \to_{\beta_{\mathsf{b}}}^+ Q^*$ does hold in λ_{b}, since:

$$P^* = \mathsf{raise}(N^*)\mathsf{box}(\lambda x.\mathsf{raise}(M^*)x^*) \to_{\beta_{\mathsf{b}}} (\lambda x.\mathsf{raise}(M^*)x^*)N^* \to_{\beta_{\mathsf{b}}} (\lambda x.\varepsilon(x)M^*)N^* = Q^* \ .$$

Some other refinements are needed, namely the identification of sub-relations in λ_{b} which allow reflection of (standard) reduction. Again, this is in the spirit of shrinking the target.

First, we introduce a new β rule

$$\mathsf{raise}(\mathsf{box}(N))\mathsf{box}(\lambda x.P) \to [N/\varepsilon(x)]P \qquad (\beta_{\mathsf{b2}}) \ ,$$

corresponding to a sequence of two β_{b}-reduction steps:

$$\mathsf{raise}(\mathsf{box}(N))\mathsf{box}(\lambda x.P) \to_{\beta_{\mathsf{b}}} (\lambda x.P)\mathsf{box}(N) \to_{\beta_{\mathsf{b}}} [N/\varepsilon(x)]P \ . \tag{8}$$

Second, we define $\Rightarrow_{\mathsf{b2}}$, a sub-relation of $\Rightarrow_{\mathsf{b}}$ in λ_{b}: in Fig. 8, replace RDX by:

$$\frac{N \to_{\mathtt{we}}^* \mathsf{box}(N') \quad M \to_{\mathtt{we}}^* \mathsf{box}(\lambda x.M') \quad [N'/\varepsilon(x)]M' \Rightarrow_{\mathsf{b2}} Q}{\mathsf{raise}(N)M \Rightarrow_{\mathsf{b2}} Q}\ RDX2$$

This is a derivable rule of $\Rightarrow_{\mathsf{b}}$: it is illuminating to see how this rule corresponds to two applications of RDX where, in each of these, the first premiss follows by reflexivity and all the action happens in the second premiss.[6]

In addition, $\Rightarrow_{\mathsf{b2}}$ determines a notion of "standard" reduction sequence that, we now argue, is standard in the official sense derived from Theorem 12 and defined right after its proof in Section 4. Rule $RDX2$ determines that the initial segment of a "standard" reduction sequence does the parallel cbb evaluation of the components of the given $\mathsf{raise}(N)M$ until a β_{b2}-redex $\mathsf{raise}(\mathsf{box}(N'))\mathsf{box}(\lambda x.M')$ emerges and is immediately reduced. Now this initial segment, which is not strictly standard (because the evaluation of N happens inside $\mathsf{raise}(\cdot)$ which is a λ), has a corresponding standard reduction sequence, namely the sequence (6).

[6] For the purpose of studying Girard's map, the following particular case of RDX, where the action happens in the *first* premiss, would have sufficed:

$$\frac{M \to_{\mathtt{we}}^* \mathsf{box}(\lambda x.M') \quad [N/\varepsilon(x)]M' \Rightarrow_{\mathsf{b}} Q}{M\mathsf{box}(N) \Rightarrow_{\mathsf{b}} Q}$$

▶ **Theorem 18** (Properties of refined Gödel's translation)**.**

1. *(Preservation of reduction) If* $M \to_{\beta_\mathtt{v}} N$ *in* $\lambda_\mathtt{v}$ *then* $M^* \to^2_{\beta_\mathtt{b}} N^*$ *in* $\lambda_\mathtt{b}$*.*
2. *(Preservation and reflection of reduction)* $M \to_{\beta_\mathtt{v}} N$ *in* $\lambda_\mathtt{v}$ *iff* $M^* \to_{\beta_{\mathtt{b2}}} N^*$ *in* $\lambda_\mathtt{b}$*.*
3. *(Preservation and reflection of evaluation)* $M \to^*_\mathtt{v} V$ *in* $\lambda_\mathtt{v}$ *iff* $M^* \to^*_\mathtt{v} V^*$ *in* $\lambda_\mathtt{b}$*.*
4. *(Preservation of standard reduction) If* $M \Rightarrow_\mathtt{v} N$ *in* $\lambda_\mathtt{v}$ *then* $M^* \Rightarrow_\mathtt{b} N^*$ *in* $\lambda_\mathtt{b}$*.*
5. *(Preservation and reflection of standard red.)* $M \Rightarrow_\mathtt{v} N$ *in* $\lambda_\mathtt{v}$ *iff* $M^* \Rightarrow_{\mathtt{b2}} N^*$ *in* $\lambda_\mathtt{b}$*.*

Proof.

Proof of 1. Follows from the "only if" half of item 2.

Proof of 2. The "only if" half is proved by induction on $M \to_{\beta_\mathtt{v}} N$. The "if" half is a consequence of two facts: (i) injectivity of $(\cdot)^*$; (ii) the image of $(\cdot)^*$ (resp. $(\cdot)^\bullet$) is closed for $\beta_{\mathtt{b2}}$-reduction and any $\beta_{\mathtt{b2}}$-reduction between images is an image of a source reduction. More precisely, the second fact is the conjunction of: (a) If $M^* \to_{\beta_{\mathtt{b2}}} P$, then there is a λ-term Q such that $Q^* = P$ and $M \to_{\beta_\mathtt{v}} Q$; (b) If $V^\bullet \to_{\beta_{\mathtt{b2}}} N$, then there is a λ-calculus value W such that $W^\bullet = N$ and $V \to_{\beta_\mathtt{v}} W$. This is proved by simultaneous induction on M and V.

Proof of 3. The result follows with the help of the following two facts:

Fact 1. If $M \to_\mathtt{v} N$ in $\lambda_\mathtt{v}$, then there exists P such that $M^* {\to_\mathtt{v}}^+ P$ and $N^* {\to_{\beta_\mathtt{a}}}^* P$ in $\lambda_\mathtt{b}$, where $\to_{\beta_\mathtt{a}}$ is *administrative* 1-step-reduction defined by closure under μ and ν of the rule

$$\mathsf{raise}(N)\mathsf{box}(M) \;\to\; MN \quad (\beta_\mathtt{a}) \;,$$

which is the $\beta_\mathtt{b}$-step $(\lambda z.\varepsilon(z)N)\mathsf{box}(M) \;\to_{\beta_\mathtt{b}}\; MN$ (for $z \notin \mathrm{FV}(N)$).[7] Notice $N^* {\to_{\beta_\mathtt{a}}}^* P$ implies $N^* {\to_\mathtt{v}}^* P$.

Fact 2. If $M^* \to^*_\mathtt{v} Q$ in $\lambda_\mathtt{b}$, then: (i) if Q is a box, then $Q = V^*$, for some value V, and $M \to^*_\mathtt{v} V$ in $\lambda_\mathtt{v}$; and (ii) if $Q = N^*$, for some N, then $M \to^*_\mathtt{v} N$ in $\lambda_\mathtt{v}$.

The "if" part of item 3 actually holds when V is replaced by an arbitrary term N, as stated in part (ii) of Fact 2. The "only if" part of item 3 follows by induction on the length of the reduction sequence. Suppose $M \to_\mathtt{v} M_0 \to^*_\mathtt{v} V$. By Fact 1 above, there exists P s.t. $M^* \to^*_\mathtt{v} P$ and $M_0^* \to^*_\mathtt{v} P$. By IH, $M_0^* \to^*_\mathtt{v} V^*$. Hence, since reduction sequences starting at M_0^* are deterministic, $P \to^*_\mathtt{v} V^*$ or $V^* \to^*_\mathtt{v} P$. As the latter is equivalent to $V^* = P$ (because V^* cannot reduce), it follows $M^* \to^*_\mathtt{v} P \to^*_\mathtt{v} V^*$, as wanted.

The proof of Fact 1 above is by induction on $M \to_\mathtt{v} N$. In the base case, one actually proves $M^* \to_\mathtt{v} N^*$, using Lemma 16. In the step case where $M = VM_0 \to_\mathtt{v} VN_0 = N$, because $M_0 \to_\mathtt{v} N_0$, we find the need for the administrative reductions from N^*.

The proof of Fact 2 above is by induction on the length of $M^* \to^*_\mathtt{v} Q$.

Proof of 4. By induction on $M \Rightarrow_\mathtt{v} N$. We spell out the case RDX. Suppose

$$\frac{M_1 \to^*_\mathtt{v} \lambda x.M_1' \quad M_2 \to^*_\mathtt{v} V \quad [V/x]M_1' \Rightarrow_\mathtt{v} N}{M_1M_2 \Rightarrow_\mathtt{v} N}\; RDX$$

We want $\mathsf{raise}(M_2^*)M_1^* \Rightarrow_\mathtt{b} N^*$. By IH and Lemma 16, $([V/x]M_1')^* = [V^\bullet/\varepsilon(x)]M_1'^* \Rightarrow_\mathtt{b} N^*$. From $M_1 \to^*_\mathtt{v} \lambda x.M_1'$ we get, by item 3 of the current theorem, $M_1^* \to^*_\mathtt{v} (\lambda x.M_1')^* = \mathsf{box}(\lambda x.M_1'^*)$. By the indifference property, $M_1^* \to^*_{\mathtt{we}} \mathsf{box}(\lambda x.M_1'^*)$. Similarly, from $M_2 \to^*_\mathtt{v} V$ we get, by item 3 of the current theorem, $M_2^* \to^*_\mathtt{v} V^* = \mathsf{box}(V^\bullet)$. By the indifference property, $M_2^* \to^*_{\mathtt{we}} \mathsf{box}(V^\bullet)$. We conclude with two applications of RDX:

$$\frac{\lambda z.\varepsilon(z)M_2^* \to^*_{\mathtt{we}} \lambda z.\varepsilon(z)M_2^* \quad M_1^* \to^*_{\mathtt{we}} \mathsf{box}(\lambda x.M_1'^*) \quad (*)}{(\lambda z.\varepsilon(z)M_2^*)M_1^* \Rightarrow_\mathtt{b} N^*}\; RDX$$

[7] The first step in the sequence (8) is administrative in this sense.

where $(*)$ is

$$\frac{\lambda x.M_1'^* \to_{\mathtt{we}}^* \lambda x.M_1'^* \quad M_2^* \to_{\mathtt{we}}^* \mathsf{box}(V^\bullet) \quad [V^\bullet/\varepsilon(x)]M_1'^* \Rightarrow_{\mathtt{b}} N^*}{(\lambda x.M_1'^*)M_2^* \Rightarrow_{\mathtt{b}} N^*}\ RDX$$

Proof of 5. The "only if" direction is proved by changing a case in the proof of item 4 of the current theorem, namely the case RDX spelled out above. Indeed, the two applications of RDX that conclude the proof may be replaced by a single application of $RDX2$.
For the "if" direction, we prove: if $M^* \Rightarrow_{\mathtt{b2}} Q$ then there is $N \in \lambda_{\mathtt{v}}$ such that $N^* = Q$ and $M \Rightarrow_{\mathtt{v}} N$. The proof is by induction on $M^* \Rightarrow_{\mathtt{b2}} Q$, and it heavily relies again on item 3 (simulation property) and the indifference property. ◀

Before extracting standardization of $\lambda_{\mathtt{v}}$ as corollary, we need the following addendum to the standardization theorem for $\lambda_{\mathtt{b}}$ (Theorem 12), concerning $\lambda_{\mathtt{b}}$-terms of the form M^*:

▶ **Theorem 19** (Addendum to standardization for $\lambda_{\mathtt{b}}$)**.** *In $\lambda_{\mathtt{b}}$, if $M^* \to_{\beta_{\mathtt{b2}}}^* N^*$ then $M^* \Rightarrow_{\mathtt{b2}} N^*$.*

Proof. The proof has the same structure as that of Theorem 12. There is a single catch, because $\Rightarrow_{\mathtt{b2}}$ is not closed under prefixing a *single* $\to_{\mathtt{we}}$-step. So the rule for $\Rightarrow_{\mathtt{b2}}$ corresponding to rule (5) in Fig. 9 cannot be stated with $\to_{\mathtt{we}}^*$, it is stated with another special binary relation $P \twoheadrightarrow_{\mathtt{we2}} P'$, inductively defined over arbitrary $\lambda_{\mathtt{b}}$-terms by closing under μ and ν the following base rule:

$$\frac{Q \to_{\mathtt{we}}^* \mathsf{box}(V) \quad P \to_{\mathtt{we}}^* \mathsf{box}(\lambda y.P')}{\mathsf{raise}(Q)P \twoheadrightarrow_{\mathtt{we2}} [V/\varepsilon(y)]P'}$$

Notice $\twoheadrightarrow_{\mathtt{we2}} \subseteq \to_{\mathtt{we}}^*$. The second derived rule in (5) shows this for the base rule. ◀

▶ **Corollary 20** (Standardization for $\lambda_{\mathtt{v}}$)**.** *In $\lambda_{\mathtt{v}}$, $M \to_{\beta_{\mathtt{v}}}^* N$ iff $M \Rightarrow_{\mathtt{v}} N$.*

Proof. The easy "if" implication follows by induction on $M \Rightarrow_{\mathtt{v}} N$. The hard "only if" implication goes via standardization for $\lambda_{\mathtt{b}}$. Suppose $M \to_{\beta_{\mathtt{v}}}^* N$. By item 2 of Theorem 18, we have $M^* \to_{\beta_{\mathtt{b2}}} N^*$. By Theorem 19, we obtain $M^* \Rightarrow_{\mathtt{b2}} N^*$ in $\lambda_{\mathtt{b}}$. Fortunately the addendum provided a statement with $\Rightarrow_{\mathtt{b2}}$ rather than $\Rightarrow_{\mathtt{b}}$, because reflection of standard reduction only works for $\Rightarrow_{\mathtt{b2}}$: item 5 of Theorem 18 concludes $M \Rightarrow_{\mathtt{v}} N$. ◀

6 Instantiations

Here we briefly illustrate two instantiations of the indeterminate comonad of $\lambda_{\mathtt{b}}$ with concrete comonads, namely the *trivial comonad* $\top \supset (\cdot)$ and the comonad ! of linear logic.

To provide a target for the trivial comonad instantiation, we add to the $\lambda_{\mathtt{v}}$-calculus a type $\top$ and a term $\star$, which we consider as a value of type $\top$. We name $\beta_{\mathtt{triv}}$ this particular case of $\beta_{\mathtt{v}}$: $(\lambda d.M)\star \to M$, with $d \notin \mathrm{FV}(M)$.

The *trivial instantiation* is defined in Fig. 10. Under this instantiation, a $\beta_{\mathtt{b}}$-reduction step in $\lambda_{\mathtt{b}}$ is simulated in this target by a $\beta_{\mathtt{v}}$-reduction step followed by a $\beta_{\mathtt{triv}}$-reduction sequence.

Composing the modal embeddings with the trivial instantiation we obtain embeddings into the considered extension of $\lambda_{\mathtt{v}}$. The resulting composition in the case of Girard's embedding is the following map $\mathcal{T} : \lambda_{\mathtt{n}} \to \lambda_{\mathtt{v}}$ (in the third clause, d is a dummy variable):

$$\mathcal{T}(x) = x\,\star \qquad \mathcal{T}(\lambda x.M) = \lambda x.\mathcal{T}(M) \qquad \mathcal{T}(MN) = \mathcal{T}(M)(\lambda d.\mathcal{T}(N))$$

$$
\begin{array}{rcl}
\mathsf{t}(X) & = & X \\
\mathsf{t}(B \supset A) & = & \mathsf{t}(B) \supset \mathsf{t}(A) \\
\mathsf{t}(\Box A) & = & \top \supset \mathsf{t}(A)
\end{array}
\qquad
\begin{array}{rcl}
\mathsf{t}(\varepsilon(x)) & = & x\star \\
\mathsf{t}(\lambda x.M) & = & \lambda x.\mathsf{t}(M) \\
\mathsf{t}(MN) & = & \mathsf{t}(M)\mathsf{t}(N) \\
\mathsf{t}(\mathsf{box}(M)) & = & \lambda d.\mathsf{t}(M) \quad (d \notin \mathrm{FV}(M))
\end{array}
$$

Figure 10 Trivial instantiation of λ_{b}.

$$
\begin{array}{rcl}
\ell(X) & = & X \\
\ell(B \supset A) & = & \ell(B) \multimap \ell(A) \\
\ell(\Box A) & = & !\ell(A)
\end{array}
\qquad
\begin{array}{rcl}
\ell(\varepsilon(x)) & = & x \\
\ell(\lambda x.M) & = & \lambda y.\mathsf{let}\,!x = y\,\mathsf{in}\,\ell(M) \\
\ell(MN) & = & \ell(M)\ell(N) \\
\ell(\mathsf{box}(M)) & = & !\ell(M)
\end{array}
$$

Figure 11 Linear instantiation of λ_{b}.

This map preserves and reflects reduction, and is found in [5], where is described as "thunk introduction implemented in Λ", using the "protecting by a λ" technique [12].

Now we turn to the second instantiation of the comonad. The *linear instantiation* of types and terms of λ_{b} into the linear λ-calculus Lin of [11] is given in Figure 11. (Recall implications of λ_{b} have a boxed type in the antecedent.)

The image of the linear instantiation may be equipped with

$$(\lambda y.\mathsf{let}\,!x = y\,\mathsf{in}\,M)(!N) \to [N/x]M \qquad (\beta_\ell)$$

which amalgamates these two reduction steps:

$$(\lambda y.\mathsf{let}\,!x = y\,\mathsf{in}\,M)(!N) \to_{\beta_\multimap} \mathsf{let}\,!x = !N\,\mathsf{in}\,M \to_{\beta_!} [N/x]$$

A fragment of Lin is thus identified, and the linear instantiation becomes an isomorphism between λ_{b} and the fragment, with β_{b} corresponding to β_ℓ. We refrain from giving more details here.

Composing the modal embeddings with the linear instantiation we obtain the embeddings shown in Fig. 12. The two compositions are translations of λ_{n} and λ_{v} into the linear λ-calculus Lin which preserve reduction. These embeddings should be compared with those in [11]. The composition with Girard's embedding (the left column in Fig. 12) gives the cbn translation in *op. cit.*, whereas the composition with Gödel's embedding (the right column in Fig. 12) gives the cbv translation in *op. cit.* except for a refinement in the clause for application: in *op. cit.* the translation is $(MN)^* = (\mathsf{let}\,!z = M^*\,\mathsf{in}\,z)N^*$.

7 Final remarks

Our conclusion is that the refined modal embeddings achieve an unification of call-by-name and call-by-value by means of the calling paradigm call-by-box. With hindsight, it is obvious that call-by-box should be enough to interpret both cbn and cbv. Call-by-box comprises these high-level ideas: boxes are distinct from values and they are *not* values; in function applications, expressions in function position reduce to values, expressions in argument position reduce to boxes; functions are only called with boxes; evaluation is weak (values do not reduce) and external (boxes do not reduce). Call-by-box can thus be the target of a compilation technique which we call **protecting-by-a-box** and that works both for cbn

$$\begin{array}{rcl} x^{\circ\ell} & = & x \\ (\lambda x.M)^{\circ\ell} & = & \lambda y.\mathsf{let}\,!x = y\,\mathsf{in}\,M^{\circ\ell} \\ (MN)^{\circ\ell} & = & M^{\circ\ell}(!N^{\circ\ell}) \end{array} \qquad \begin{array}{rcl} V^{*\ell} & = & !V^{\bullet\ell} \\ (MN)^{*\ell} & = & (\lambda y.\mathsf{let}\,!z = y\,\mathsf{in}\,zN^{*\ell})M^{*\ell} \\ x^{\bullet\ell} & = & x \\ (\lambda x.M)^{\bullet\ell} & = & \lambda y.\mathsf{let}\,!x = y\,\mathsf{in}\,M^{*\ell} \end{array}$$

Figure 12 Compositions of the modal embeddings and the linear instantiation.

and cbv: cbn is obtained by protecting arguments with boxes (the old idea of freezing the argument to delay its evaluation); cbv is obtained by restricting boxes to boxed values (hence functions are called with values only) and always having values wrapped as boxes (enabling the calling of functions with values).

Notice this is a story with a cbn side and a cbv side. The cbn side is an abstraction of the protecting-by-a-λ technique, as shown through the trivial instantiation. The cbn side is also what the literature offers. Hatcliff and Danvy [5, 6] formalized an abstract version of protecting-by-a-λ as the thunk-introduction map from cbn λ-calculus into the λ-calculus with thunks (Λ_τ), and a separate variant of the thunk-introduction map directly into the λ-calculus. The former corresponds to our Girard's embedding, while the latter corresponds to the composition $\mathcal{T}$ of Section 6. But even here our results offer some improvements. First, we observed that $\mathcal{T}$ is connected to Girard's embedding through the trivial instantiation. Second, the precise formulation of the target system of Girard's embedding is important. Our care in formulating $\lambda_\Box$ so that it is closed under substitution and enjoys subject reduction is not matched in the treatment of typed Λ_τ [6]. And then we went further from $\lambda_\Box$ to λ_{b}, and this alone improved the properties of Girard's embedding as witnessed in Theorem 14.

The connection between calling paradigms and embeddings of intuitionistic logic into linear logic [4, 7, 9, 10] has its full treatment in [11]. Regarding cbn and cbv, our results match the results of [11], but in a more abstract and simpler setting - in fact we go further, since we treat proper standardization, and that is a key ingredient in our claim that cbn and cbv are unified by cbb. In obtaining results through embeddings into modal logic similar to those through embedding into linear logic, one sees that already modality, without the need for linearity, brings the calling mechanisms into the scope of the Curry-Howard isomorphism.

Inspired by linear logic, the bang-calculus [3] was recently proposed as a "generalization" of cbn and cbv. The part of *op. cit.* not concerned with denotational semantics compares with the initial part of the present paper, up to Section 3, where we dealt with unrefined target and embeddings (but notice the conceptual difference: for us, boxes are not values); all the work that comes after Section 3, about the refined target and embeddings, and which is the core of our contribution, is beyond the scope of [3].

As to future work, we would like to deepen the study of instantiations, both by considering other instantiations, and by investigating whether one obtains, through the composition of instantiations with embeddings, not only known maps, but also their properties.

References

1 H.P. Barendregt. *The Lambda Calculus*. North-Holland, 1984.

2 G. M. Bierman and V. C. V. de Paiva. On an intuitionistic modal logic. *Studia Logica*, 65:383–416, 2000.

3 T. Ehrhard and G. Guerrieri. The bang calculus: an untyped lambda-calculus generalizing call-by-name and call-by-value. In *Proc. of PPDP '16*, pages 174–187. ACM, 2016.

4 J.-Y. Girard. Linear logic. *Theor. Comput. Sci.*, 50(1):1–102, 1987.
5 J. Hatcliff and O. Danvy. Thunks and the λ-calculus. *J. Funct. Program.*, 7(3):303–319, 1997.
6 J. Hatcliff and O. Danvy. Thunks and the λ-Calculus (Extended Version). Technical Report BRICS RS-97-7, DIKU, 1997.
7 Y. Lafont. From proof-nets to interaction nets. In J. Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic*, pages 225–247. Cambridge Univ. Press, 1995.
8 Ralph Loader. Notes on simply typed lambda calculus. Technical Report ECS-LFCS-98-381, LFCS, Univ. of Edinburgh, 1998.
9 I. Mackie. *The Geometry of Implementation.* PhD thesis, Imperial College, 1994.
10 I. Mackie. Encoding strategies in the lambda calculus with interaction nets. In *Revised Selected Papers from IFL 2005*, volume 4015 of *Lect. Notes in Comput. Sci.*, pages 19–36. Springer, 2006.
11 J. Maraist, M. Odersky, D. N. Turner, and P. Wadler. Call-by-name, call-by-value, call-by-need and the linear lambda calculus. *Theor. Comput. Sci.*, 228(1-2):175–210, 1999.
12 G. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theor. Comput. Sci.*, 1:125–159, 1975.
13 D. Prawitz. *Natural Deduction.* Almquist and Wiksell, Stockholm, 1965.
14 A. Sabry and P. Wadler. A reflection on call-by-value. *ACM Trans. Program. Lang. Syst.*, 19(6):916–941, 1997.
15 A. Troelstra. Introductory note to 1933f. In *Kurt Gödel Collected Works*, pages 296–299. Oxford Univ. Press, 1986.
16 A. Troelstra and H. Schwichtenberg. *Basic Proof Theory.* Cambridge Univ. Press, 2000.

A Natural deduction system for the logic IS4

The formulae of the logic **IS4** are given by the grammar

$$\begin{array}{lcl} A & ::= & X \mid A \supset A \mid B \\ B & ::= & \Box A \end{array}$$

Formulae of the form B are called boxed formulae. Note that the antecedent of an implication need not be boxed.

Sequents of the natural deduction system are $\Gamma \vdash A$ where the antecedent Γ is a multiset of formulae, again not necessarily boxed. The proof rules are

$$\frac{}{\Gamma, A \vdash A} \qquad \frac{\Gamma, A_1 \vdash A_2}{\Gamma \vdash A_1 \supset A_2} \qquad \frac{\Gamma \vdash A_1 \supset A_2 \quad \Gamma \vdash A_1}{\Gamma \vdash A_2}$$

$$\frac{\Gamma \vdash B_1 \quad \Gamma \vdash B_n \quad B_1, \ldots, B_n \vdash A}{\Gamma \vdash \Box A} \qquad \frac{\Gamma \vdash \Box A}{\Gamma \vdash A}$$

Note that the $B_1, \ldots, B_n$ in the $\Box$ introduction rule are boxed formulae.

In the corresponding term calculus, terms are given by the grammar

$$M, N ::= x \mid \lambda x.M \mid MN \mid \mathsf{cobind}(M_1, \ldots, M_n, x_1 \ldots x_n.N) \mid \varepsilon(M)$$

where for the term form $\mathsf{cobind}(M_1, \ldots, M_n, x_1 \ldots x_n.N)$ it is required that $\mathrm{FV}(N) \subseteq \{x_1, \ldots, x_n\}$.

Typing judgments are $\Gamma \vdash M : A$ where the antecedent Γ is a set of type assignments $x : A$ with all x distinct. The typing rules are

$$\frac{}{\Gamma, x : A \vdash x : A} \qquad \frac{\Gamma, x : A_1 \vdash M : A_2}{\Gamma \vdash \lambda x.M : A_1 \supset A_2} \qquad \frac{\Gamma \vdash M : A_1 \supset A_2 \quad \Gamma \vdash N : A_1}{\Gamma \vdash MN : A_2}$$

$$\frac{\Gamma \vdash M_1 : B_1 \quad \Gamma \vdash M_n : B_n \quad x_1 : B_1, \ldots, x_n : B_n \vdash N : A}{\Gamma \vdash \mathsf{cobind}(M_1, \ldots, M_n, x_1 \ldots x_n.N) : \Box A} \qquad \frac{\Gamma \vdash M : \Box A}{\Gamma \vdash \varepsilon(M) : A}$$

The reduction relation between terms is given by the axioms

$$\frac{}{(\lambda x.M)N \to [N/x]M}\ \beta_{\supset}$$

$$\frac{}{\varepsilon(\mathsf{cobind}(M_1,\ldots,M_n,x_1\ldots x_n.N)) \to [M_1/x_1,\ldots,M_n/x_n]N}\ \beta_{\Box}$$

together with the rules of compatible closure.

In the categorical semantics of the natural deduction system of **IS4** in terms of a cartesian closed category with a lax monoidal comonad, ε corresponds to the counit of the comonad and cobind to a combination of the comultiplication and the lax monoidality laws.

▶ **Proposition 2.**
1. *For all* Γ, A *in* $\lambda_{\Box}$*,* $\Gamma \vdash M : A$ *in* $\lambda_{\Box}$ *iff* $\Gamma \vdash M^{\sharp}_{|\Gamma|} : A$ *in the term calculus for* **IS4**.
2. *For all* U *s. t.* $\mathrm{FV}(M) \subseteq U$*,* $M \to N$ *in* $\lambda_{\Box}$ *iff* $M^{\sharp}_{U} \to N^{\sharp}_{U}$ *in the term calculus for* **IS4**.

Proof.
1. The "only if" direction is proved by induction on the given $\lambda_{\Box}$ typing derivation. The case for the typing rule of box goes through in **IS4** using the $\Box$ introduction rule, where the first n premises are axioms (one for each of the n formulas in the context). The "if" direction is by induction on M. In the case $M = \mathsf{box}(N)$, assuming $\Gamma = x_1 : B_1,\ldots,x_n : B_n$, $M^{\sharp}_{|\Gamma|} = \mathsf{cobind}(x_1,\ldots,x_n,x_1\ldots x_n.N^{\sharp}_{|\Gamma|})$ and the hypothesis implies $A = \Box A_0$ and $\Gamma \vdash N^{\sharp}_{|\Gamma|} : A_0$ in $\lambda_{\Box}$ (for some A_0), so the IH and the $\Box$ introduction rule of $\lambda_{\Box}$ can be used to conclude.
2. Routine induction on the given reduction derivation in both directions. ◀

▶ **Proposition 3.** *For all* Γ, A *in* $\lambda_{\Box}$*, if* $\Gamma \vdash M : A$ *in the term calculus for* **IS4***, then there exists* N *such that* $\Gamma \vdash N : A$ *in* $\lambda_{\Box}$*.*

Proof. By induction on the given **IS4** typing derivation. The case for the $\Box$ introduction rule needs admissibility of both weakening and the typing rule for substitution. ◀

Probabilistic Rewriting: Normalization, Termination, and Unique Normal Forms

Claudia Faggian
Université de Paris, IRIF, CNRS, F-75013 Paris, France
faggian@irif.fr

Abstract

While a mature body of work supports the study of rewriting systems, abstract tools for Probabilistic Rewriting are still limited. We study in this setting questions such as uniqueness of the result (unique limit distribution) and normalizing strategies (is there a strategy to find a result with greatest probability?). The goal is to have tools to analyse the operational properties of *probabilistic* calculi (such as probabilistic lambda-calculi) whose evaluation is also non-deterministic, in the sense that different reductions are possible.

2012 ACM Subject Classification Theory of computation → Probabilistic computation; Theory of computation → Rewrite systems; Theory of computation → Logic

Keywords and phrases probabilistic rewriting, PARS, abstract rewriting systems, confluence, probabilistic lambda calculus

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.19

Related Version `http://arxiv.org/abs/1804.05578`

Acknowledgements This work benefitted of fruitful discussions with U. Dal Lago, B. Valiron, and T. Leventis. I also wish to thank the anonymous referees for valuable comments and suggestions.

1 Introduction

Rewriting Theory [39] is a foundational theory of computing. Its impact extends to both the theoretical side of computer science, and the development of programming languages. A clear example of both aspects is the paradigmatic term rewriting system, λ-calculus, which is also the foundation of functional programming. *Abstract Rewriting Systems (ARS)* are the general theory which captures the common substratum of rewriting theory, independently of the particular structure of the objects. It studies properties of terms transformations, such as normalization, termination, unique normal form, and the relations among them. Such results are a powerful set of tools which can be used when we study the computational and operational properties of any calculus or programming language. Furthermore, the theory provides tools to study and compare strategies, which become extremely important when a system *may* have reductions leading to a normal form, but *not necessarily*. Here we need to know: is there a strategy which is guaranteed to lead to a normal form, if any exists (*normalizing* strategies)? Which strategies diverge if at all possible (*perpetual* strategies)?

Probabilistic Computation models uncertainty. Probabilistic models such as automata [34], Turing machines [37], and the λ-calculus [36] exist since long. The pervasive role it is assuming in areas as diverse as robotics, machine learning, natural language processing, has stimulated the research on probabilistic programming languages, including functional languages [27, 35, 32] whose development is increasingly active. A typical programming language supports at least discrete distributions by providing a probabilistic construct which models sampling from a distribution. This is also the most concrete way to endow the λ-calculus with probabilistic choice [13, 10, 16]. Within the vast research on models of probabilistic systems, we wish to mention that probabilistic rewriting is the explicit base of PMaude [1], a language for specifying probabilistic concurrent systems.

© Claudia Faggian;
licensed under Creative Commons License CC-BY
4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019).
Editor: Herman Geuvers; Article No. 19; pp. 19:1–19:25
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Probabilistic Rewriting. Somehow surprisingly, while a large and mature body of work supports the study of rewriting systems – even infinitary ones [12, 24] – work on the abstract theory of *probabilistic* rewriting systems is still sparse. The notion of *Probabilistic* Abstract Reduction Systems (PARS) has been introduced by Bournez and Kirchner in [5], and then extended in [4] to account for non-determinism. Recent work [7, 15, 25, 3] shows an increased research interest. The key element in *probabilistic* rewriting is that even when the probability that a term leads to a normal form is 1 (*almost sure termination*), that degree of certitude is typically not reached in any finite number of steps, but it appears as a limit. Think of a rewrite rule (as in Fig. 1) which rewrites c to either the value $\mathtt{T}$ or c, with equal probability 1/2. We write this $c \to \{c^{1/2}, \mathtt{T}^{1/2}\}$. After n steps, c reduces to $\mathtt{T}$ with probability $\frac{1}{2} + \frac{1}{2^2} + \cdots + \frac{1}{2^n}$. Only at the limit this computation terminates with probability 1 .

The most well-developed literature on PARS is concerned with methods to prove almost sure termination, see e.g. [4, 19, 3] (this interest matches the fact that there is a growing body of methods to establish AST [2, 20, 22, 30]). However, considering rewrite rules subject to probabilities opens numerous other questions on PARS, which motivate our investigation.

We study a rewrite relation on distributions, which describes the evolution of a probabilistic system, for example a probabilistic program P. The *result* of the computation is a distribution β over all the possible values of P. The intuition (see [27]) is that the program P is executed, and random choices are made by sampling. This process eventually defines a distribution β over the various outputs that the program can produce. We write this $P \overset{\infty}{\Rrightarrow} \beta$.

What happens if the *evaluation* of a term P is also *non-deterministic*? Remember that non-determinism arises naturally in the λ-calculus, because a term may have several redexes. This aspect has practical relevance to programming. Together with the fact that the result of a terminating computation is unique, it is key to the inherent parallelism of functional programs (see *e.g.* [29]). When assuming non-deterministic evaluation, several questions on PARS arise naturally. For example: (1.) when is the result unique? (naively, if $P \overset{\infty}{\Rrightarrow} \alpha$ and $P \overset{\infty}{\Rrightarrow} \beta$, is $\alpha = \beta$?) (2.) Do all rewrite sequences from the same term have the same probability to reach a result? (3.) If not, does there exist a strategy to find a result with greatest probability?

Such questions are relevant not only to the theory, but also to the practice of computing. We believe that to study them, we can advantageously adapt techniques from Rewrite Theory. However, we *cannot assume that standard properties of ARS hold for PARS.* The game-changer is that termination appears as a *limit.* In Sec. 4.4 we show that a well-known ARS property, Newman's Lemma, does not hold for PARS. This is not surprising; indeed, Newman's Lemma is known not to hold in general for infinitary rewriting [23, 26]. Still, our counter-example points out that moving from ARS to PARS is non-trivial. There are two main issues: we need to find the *right formulation* and the *right proof technique.* It seems especially important to have a collection of proof methods which apply well to PARS.

Content and contributions. Probability is concerned with *asymptotic* behaviour: what happens not after a finite number n of steps, but *when n tends to infinity.* In this paper we focus on the asymptotic behaviour of rewrite sequences *with respect to normal forms.* We study computational properties such as (1.),(2.),(3.) above. We do so with the point of view of ARS, aiming for properties which *hold independently* of the specific nature of the rewritten objects; the purpose is to have tools which apply to any probabilistic rewriting system.

After introducing and motivating our formalism (Sec. 2 and 3), in Sec. 4, we extend to the probabilistic setting the notions of *Normalization (WN), Termination (SN) and Unique Normal Form* (UN). In the rest of the paper, we provide methods and criteria to establish

these properties, and we uncover relations between them. In particular, we study *normalizing strategies*. To do so, we extend to the probabilistic setting a proposal by Van Oostrom [40], which is based on Newman's property of Random Descent [31, 40, 41] (see Sec. 1.1). The Random Descent method turns out to provide proof techniques which are well suited to PARS. Specific contributions are the following.

- We propose an analogue of `UN` for PARS. This is not obvious; the question was already studied in [15] for PARS which are `AST`, but their solution does not extend to general PARS.
- We investigate the classical ARS method to prove `UN` via *confluence*. It turns out that the notion of confluence does not need to be as strong as the classical case would suggest, broadening its scope of application. Subtle aspects appear when dealing with limits, and the proof demand specific techniques.
- We develop a probabilistic extension of the ARS notions of Random Descent ($\mathcal{E}$-RD, Sec. 5) and of being *better* ($\mathcal{R}$-better, Sec. 7) as tools to analyze and compare strategies, in analogy to their counterpart in [40]. Both properties are here parametric with respect to a chosen event of interest. $\mathcal{E}$-RD entails that all rewrite sequences from a term lead to the *same result*, in the *same expected number of steps* (the average of number of steps, weighted w.r.t. probability). $\mathcal{R}$-better offers a method to compare strategies ("strategy $\mathcal{S}$ is always better than strategy $\mathcal{T}$") w.r.t. the *probability* of reaching a result and the *expected time* to reach a result. It provides a sufficient criterion to establish that a strategy is *normalizing* (resp. *perpetual*) *i.e.* the strategy is guaranteed to lead to a result with maximal (resp. minimal) probability. A significant technical feature (inherited from [40]) is that both notions of $\mathcal{E}$-RD and $\mathcal{R}$-better come with a characterization via a *local condition* (in ARS, a typical example of a local vs global condition is local confluence vs confluence).

We apply these methods to study a probabilistic λ-calculus, which we discuss below together with the notion of Random Descent. A deeper example of application to probabilistic λ-calculus is in [18]; we discuss it in Sec.8 "Further work and applications".

▶ Remark (On the term *Random* Descent). Please note that in [31], the term *Random* refers to non-determinism (in the choice of the redex), *not to randomized* choice.

Related work. We discuss related work in the context of PARS [4, 5]. We are not aware of any work which investigates *normalizing strategies* (or *normalization* in general, rather than termination). Instead, *confluence* in probabilistic rewriting has already drawn interesting work. A notion of confluence for a probabilistic rewrite system defined over a λ-calculus is studied in [14, 9]; in both case, the probabilistic behavior corresponds to measurement in a quantum system. The work more closely related to our goals is [15]. It studies confluence of non-deterministic PARS in the case of finitary termination (being finitary is the reason why a Newman's Lemma holds), and in the case of `AST`. As we observe in Sec. 4.3, their notion of unique limit distribution (if α, β are limits, then $\alpha = \beta$), while simple, it is not an analogue of `UN` for general PARS; we extend the analysis beyond `AST`, to the general case, which arises naturally when considering probabilistic λ-calculus. On confluence, we also mention [25], whose results however do not cover *non-deterministic PARS*; the probability of the limit distribution is concentrated in a single element, in the spirit of Las Vegas Algorithms. [25] revisits results from [5], while we are in the non-deterministic framework of [4].

The way we define the *evolution of PARS*, via the one-step relation $\rightrightarrows$, follows the approach in [7], which also contains an embryo of the current work (a form of diamond property); the other results and developments are novel. A technical difference with [7] is

that for the formalism to be general, a refinement is necessary (see Sec. 2.2); the issue was first pointed out in [15]. Our refinement is a variation of the one introduced (for the same reasons) in [3]; we however do not strictly adopt it, because we prefer to use a standard definition of distribution. [3] demonstrates the equivalence with the approach in [4].

1.1 Key notions

Random Descent. Newman's Random Descent (RD) [31] is an ARS property which guarantees that normalization suffices to establish both termination and uniqueness of normal forms. Precisely, if an ARS has random descent, paths to a normal form do not need to be unique, but they have *unique length.* In its essence: *if a normal form exists, all rewrite sequences lead to it, and all have the* same length[1]. While only few systems directly verify it, RD is a powerful ARS tool; a typical use in the literature is to prove that *a strategy* has RD, to conclude that it is *normalizing.* A well-known property which implies RD is a form of diamond: "$\leftarrow \cdot \rightarrow \subseteq (\rightarrow \cdot \leftarrow) \cup =$".

In [40] Von Oostrom defines a characterization of RD by means of a *local* property and proposes RD as a uniform method to (locally) compare strategies for normalization and minimality (resp. perpetuality and maximality). [41] extends the method and abstracts the notion of length into a notion of measure. In Sec. 5 and 7 we develop similar methods in a *probabilistic* setting. The analogous of *length*, is the *expected number of steps* (Sec. 5.1).

Probabilistic Weak λ-calculus. A notable example of system which satisfies RD is the pure untyped λ-calculus endowed with call-by-value (CbV) weak evaluation. *Weak* [21, 6] means that reduction does not evaluate function bodies (*i.e.* the scope of λ-abstractions). We recall that weak CbV is the basis of the ML/CAML family of functional languages (and of most probabilistic functional languages). Because of RD, weak CbV λ-calculus has *striking properties* (see e.g. [8] for an account). First, if a term M has a normal form N, any rewrite sequence will find it; second, the number n of steps such that $M \rightarrow^n N$ is always the same.

In Sec. 6, we study a probabilistic extension of weak CbV, $\Lambda_{\oplus}^{\mathtt{weak}}$. We show that it has analogous properties to its classical counterpart: all rewrite sequences converge to the same result, in the same *expected* number of steps.

Local vs global conditions. To work *locally* means to reduce a test problem which is global, *i.e.*, quantified over *all rewrite sequences* from a term, to local properties (quantified only over *one-step reductions* from the term), thus reducing the space of search when testing.

A paradigmatic example of a global property is confluence (CR: $b\ {}^*\!\!\leftarrow a \rightarrow^* c \Rightarrow \exists d$ s.t. $b \rightarrow^* d\ {}^*\!\!\leftarrow c$). Its global nature makes it difficult to establish. A standard way to factorize the problem is: (1.) prove termination and (2.) prove *local* confluence (WCR: $b \leftarrow a \rightarrow c \Rightarrow \exists d$ s.t. $b \rightarrow^* d\ {}^*\!\!\leftarrow c$). This is exactly *Newman's lemma: Termination + WCR ⇒ CR.* The beauty of Newman's lemma is that a global property (CR) is guaranteed by a local property (WCR). Locality is also the strength and beauty of the RD method. While Newman's lemma fails in a probabilistic setting (see Sec. 4.4), RD methods can be adapted (Sec. 5 and 7).

[1] or, in Newman's original terminology: the end-form is reached by *random descent* (whenever $x \rightarrow^k y$ and $x \rightarrow^n u$ with u in normal form, all maximal reductions from y have length $n - k$ and end in u).

1.2 Probabilistic λ-calculus and (Non-)Unique Result

Rewrite theory provides numerous tools to study uniqueness of normal forms, as well as techniques to study and compare strategies. This is not the case in the probabilistic setting. Perhaps a reason is that when extending the λ-calculus with a choice operator, confluence is lost, as was observed early [11]; we illustrate it in Example 1.1 and 1.2, which is adapted from [11, 10]. The way to deal with this issue in probabilistic λ-calculi (e.g. [13, 10, 16]) has been to *fix a deterministic reduction strategy*, typically "leftmost-outermost". To fix a strategy is not satisfactory, neither for the theory nor the practice of computing. To understand why this matters, recall for example that confluence of the λ-calculus is what makes functional programs inherently parallel: every sub-expression can be evaluated in parallel, still, we can reason on a program using a deterministic sequential model, because the result of the computation is independent of the evaluation order (we refer to [29], and to Harper's text "Parallelism is not Concurrency" for discussion on *deterministic* parallelism, and how it differs from concurrency). Let us see what happens in the probabilistic case.

▶ **Example 1.1** (Confluence failure)**.** Let us consider the untyped λ-calculus extended with a binary operator $\oplus$ which models probabilistic choice. Here $\oplus$ is just flipping a fair coin: $M \oplus N$ reduces to either M or N with equal probability $1/2$; we write this as $M \oplus N \to \{M^{\frac{1}{2}}, N^{\frac{1}{2}}\}$.

Consider the term PQ, where $P = (\lambda x.x)(\lambda x.x \texttt{ XOR } x)$ and $Q = (\texttt{T} \oplus \texttt{F})$; here `XOR` is the standard constructs for the exclusive `OR`, `T` and `F` are terms which code the booleans.

- If we evaluate P and Q independently, from P we obtain $\lambda x.(x \texttt{ XOR } x)$, while from Q we have either `T` or `F`, with equal probability $1/2$. By composing the partial results, we obtain $\{(\texttt{T XOR T})^{\frac{1}{2}}, (\texttt{F XOR F})^{\frac{1}{2}}\}$, and therefore $\{\texttt{F}^1\}$.
- If we evaluate PQ sequentially, in a standard left-most outer-most fashion, PQ reduces to $(\lambda x.x \texttt{ XOR } x)Q$ which reduces to $(\texttt{T} \oplus \texttt{F}) \texttt{ XOR } (\texttt{T} \oplus \texttt{F})$ and eventually to $\{\texttt{T}^{\frac{1}{2}}, \texttt{F}^{\frac{1}{2}}\}$.

▶ **Example 1.2.** The situation becomes even more complex if we examine also the possibility of diverging; try the same experiment as above on the term PR, with $R = (\texttt{T} \oplus \texttt{F}) \oplus \Delta\Delta$ (where $\Delta = \lambda x.xx$). Proceeding as before, we now obtain either $\{\texttt{F}^{\frac{1}{2}}\}$ or $\{\texttt{T}^{\frac{1}{8}}, \texttt{F}^{\frac{1}{8}}\}$.

We do not need to loose the features of λ-calculus in the *probabilistic* setting. In fact, while some care is needed, determinism of the evaluation *can be relaxed* without giving up uniqueness of the result: the calculus we introduce in Sec. 6 is an example (we relax determinism to RD); we fully develop this direction in further work [18]. To be able to do so, we *need abstract tools and proof techniques* to analyze *probabilistic* rewriting. The same need for theoretical tools holds, more in general, whenever we desire to have a probabilistic language which allows for *deterministic parallel reduction*.

In this paper we focus on *uniqueness of the result*, rather than confluence, which is an important and sufficient, but not necessary property.

2 Probabilistic Abstract Rewriting System

We assume the reader familiar with the basic notions of rewrite theory (such as Ch. 1 of [39]), and of *discrete* probability theory. We review the basic language of both. We then recall the definition of PARS from [5, 4], and explain on examples how a system described by a PARS evolves. This will motivate the formalism which we introduce in Sec. 3.

Basics on ARS. An *abstract rewrite system (ARS)* is a pair $\mathcal{C} = (C, \to)$ consisting of a set C and a binary relation $\to$ on C; $\to^*$ denotes the transitive reflexive closure of $\to$. An element $u \in C$ is in **normal form** if there is no c with $u \to c$; $\texttt{NF}_{\mathcal{C}}$ denotes the set of the normal forms

of $\mathcal{C}$. If $c \to^* u$ and $u \in \mathtt{NF}_\mathcal{C}$, we say c has a normal form u. $\mathcal{C}$ has the property of **unique normal form (*with respect to reduction*)**(UN) if $\forall u, v \in \mathtt{NF}_\mathcal{C}, \big(c \to^* u \;\&\; c \to^* v \Rightarrow u = v\big)$. $\mathcal{C}$ has the **normal form property** (NFP) if $\forall b, c \in C, \forall u \in \mathtt{NF}_\mathcal{C}, \big(b \to^* c \;\&\; b \to^* u \Rightarrow c \to^* u\big)$. NFP implies UN. The fact that an ARS has unique normal forms implies neither that all terms have a normal form, nor that if a term has a normal form, each rewrite sequence converges to it. A term c is **terminating**[2] (aka **strongly normalizing**, SN), if it has no infinite sequence $c \to c_1 \to c_2 ...$; it is **normalizing** (aka **weakly normalizing**, WN), if it has a normal form. These are all important properties to establish about an ARS, as it is important to have a rewrite strategy which finds a normal form, if it exists.

Basics on Probabilities. The intuition is that random phenomena are observed by means of experiments (running a probabilistic program is such an experiment); each experiment results in an outcome. The collection of all possible outcomes is represented by a set, called the **sample space** Ω. When the sample space Ω is *countable*, the theory is simple. A *discrete probability space* is given by a pair (Ω, μ), where Ω is a *countable* set, and μ is a **discrete probability distribution** on Ω, *i.e.* a function $\mu : \Omega \to [0, 1]$ such that $\sum_{\omega \in \Omega} \mu(\omega) = 1$. A probability measure is assigned to any subset $A \subseteq \Omega$ as $\mu(\mathcal{A}) = \sum_{\omega \in A} \mu(\omega)$. In the language of probabilists, a subset of Ω is called an *event*.

▶ **Example 2.1** (Die). Consider tossing a die once. The space of possible outcomes is the set $\Omega = \{1, 2, 3, 4, 5, 6\}$. The probability μ of each outcome is $1/6$. The event *"result is odd"* is the subset $A = \{1, 3, 5\}$, whose probability is $\mu(A) = 1/2$.

Each *function* $F : \Omega \to \Delta$, where Δ is another countable set, **induces a probability distribution** μ^F on Δ by composition: $\mu^F(d) := \mu(F^{-1}(d))$ *i.e.* $\mu\{\omega \in \Omega : F(\omega) = d\}$. Thus (Δ, μ^F) is also a probability space. In the language of probability theory, F is called a *discrete random variable* on (Ω, μ). The **expected value** (also called the expectation or mean) of a random variable F is the weighted (in proportion to probability) average of the possible values of F. Assume $F : \Omega \to \Delta$ discrete and $g : \Delta \to \mathbb{R}$ a non-negative function, then $E(g(F)) = \sum_{d \in \Delta} g(d)\mu_F(d)$.

(Sub)distributions: operations and notation. We need the notion of subdistribution to account for unsuccessful computations and partial results. Given a countable set Ω, a function $\mu : \Omega \to [0, 1]$ is a probability **subdistribution** if $\|\mu\| := \sum_{\omega \in \Omega} \mu(\omega) \leq 1$. We write $\mathtt{DST}(\Omega)$ for the set of subdistributions on Ω. With a slight abuse of language, we often use the term distribution also for subdistribution. The *support* of μ is the set $\mathtt{Supp}(\mu) = \{a \in \Omega \mid \mu(a) > 0\}$. $\mathtt{DST}^\mathtt{F}(\Omega)$ denotes the set of $\mu \in \mathtt{DST}(\Omega)$ with *finite support*.

$\mathtt{DST}(\Omega)$ is equipped with the **order relation** of functions : $\mu \leq \rho$ if $\mu(a) \leq \rho(a)$ for each $a \in \Omega$. **Multiplication** for a scalar $(p \cdot \mu)$ and **sum** $(\sigma + \rho)$ are defined as usual, $(p \cdot \mu)(a) = p \cdot \mu(a)$, $(\sigma + \rho)(a) = \sigma(a) + \rho(a)$, provided $p \in [0, 1]$, and $\|\sigma\| + \|\rho\| \leq 1$.

We adopt the following **convention**: if $\Omega' \subseteq \Omega$, and $\mu \in \mathtt{DST}(\Omega')$, we also write $\mu \in \mathtt{DST}(\Omega)$, with the implicit assumption that the extension behaves as μ on Ω', and is 0 otherwise. In particular, we identify a subdistribution and its support.

---

[2] Please observe that the *terminology is community-dependent*. In logic: Strong Normalization, Weak Normalization, Church-Rosser (hence the *standard abbreviations* SN, WN, CR). In computer science: Termination, Normalization, Confluence.

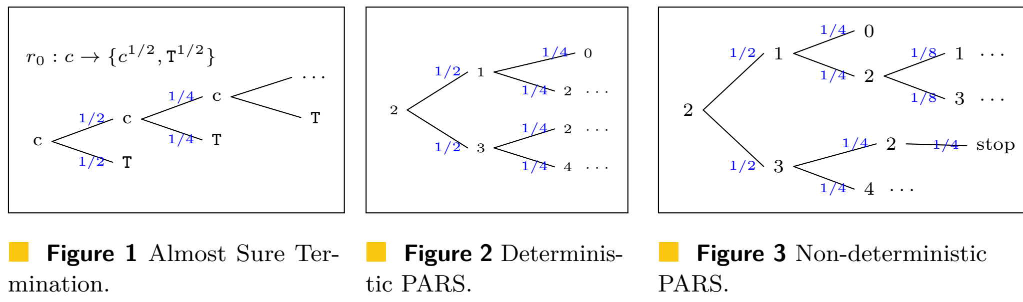


Figure 1 Almost Sure Termination.

Figure 2 Deterministic PARS.

Figure 3 Non-deterministic PARS.

▶ **Notation 2.2** (Representation). *We represent a (sub)distribution by explicitly indicating the support, and (as superscript) the probability assigned to each element by μ. We write $\mu = \{a_0^{p_0}, \dots, a_n^{p_n}\}$ if $\mu(a_0) = p_0, \dots, \mu(a_n) = p_n$ and $\mu(a_j) = 0$ otherwise.*

2.1 Probabilistic Abstract Rewrite Systems (PARS)

A *probabilistic abstract rewrite system (PARS)* is a pair $\mathcal{A} = (A, \to)$ of a countable set A and a relation $\to \subseteq A \times \mathtt{DST}^{\mathtt{F}}(A)$ such that for each $(a, \beta) \in \to$, $\|\beta\| = 1$. We write $a \to \beta$ for $(a, \beta) \in \to$ and we call it a *rewrite step*, or a *reduction*. An element $a \in A$ is in *normal form* if there is no β with $a \to \beta$. We denote by $\mathtt{NF}_{\mathcal{A}}$ the set of the normal forms of $\mathcal{A}$ (or simply $\mathtt{NF}$ when $\mathcal{A}$ is clear). A PARS is *deterministic* if, for all a, there is at most one β with $a \to \beta$.

▶ Remark. The intuition behind $a \to \beta$ is that the rewrite step $a \to b$ ($b \in A$) has probability $\beta(b)$. The total probability given by the sum of all steps $a \to b$ is 1.

Probabilistic vs Non-deterministic. It is important to have clear the distinction between probabilistic choice (which *globally happens with certitude*) and non-deterministic choice (which leads to different distributions of outcomes.) Let us discuss some examples.

▶ **Example 2.3** (A deterministic PARS). Fig. 2 shows a simple random walk over $\mathbb{N}$, which describes a gambler starting with 2 points and playing a game where every time he either gains 1 point with probablity 1/2 or looses 1 point with probability 1/2. This system is encoded by the following PARS on $\mathbb{N}$: $n + 1 \to \{n^{1/2}, (n + 2)^{1/2}\}$. Such a PARS is *deterministic*, because for every element, at most one choice applies. Note that 0 is a normal form.

▶ **Example 2.4** (A non-deterministic PARS). Assume now (Fig. 3) that the gambler of Example 2.3 is also given the possibility to stop at any time. The two choices are here encoded as follows: $n + 1 \to \{n^{1/2}, (n + 2)^{1/2}\}$, $\quad n + 1 \to \{\mathtt{stop}^1\}$.

2.2 Evolution of a system described by a PARS

We now need to explain how a system which is described by a PARS evolves. An option is to follow the stochastic evolution of a single run, *a sampling at a time*, as we have done in Fig. 1, 2, and 3. This is the approach in [4], where non-determinism is solved by the use of policies. Here we follow a different (though equivalent) way (see the Related Work Section). We describe the possible states of the system, at a certain time t, *globally*, as a distribution on the space of all terms. The evolution of the system is then a sequence of distributions. Since all the probabilistic choices are taken together, the only source of choice in the evolution is non-determinism. This global approach allows us to deal with non-determinism by using techniques which have been developed in Rewrite Theory. Before introducing the formal definitions, we informally examine some examples, and point out why some care is needed.

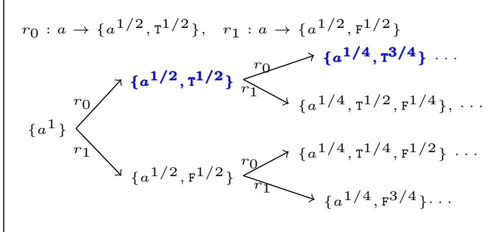


Figure 4 Ex.2.6 (non-deterministic PARS).

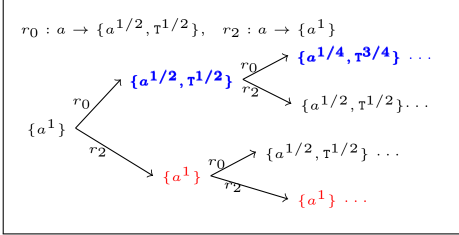


Figure 5 Ex.2.7 (non-deterministic PARS).

▶ **Example 2.5** (Fig.1 continued)**.** The PARS described by the rule $r_0 : c \to \{c^{1/2}, \mathtt{T}^{1/2}\}$ (in Fig. 1) evolves as follows: $\{c\}, \{c^{1/2}, \mathtt{T}^{1/2}\}, \{c^{1/4}, \mathtt{T}^{3/4}\}, \dots$.

▶ **Example 2.6** (Fig.4)**.** Fig. 4 illustrates the possible evolutions of a non-deterministic system which has two rules: $r_0 : a \to \{a^{1/2}, \mathtt{T}^{1/2}\}$ and $r_1 : a \to \{a^{1/2}, \mathtt{F}^{1/2}\}$. The arrows are annotated with the chosen rule.

▶ **Example 2.7** (Fig.5)**.** Fig. 5 illustrates the possible evolutions of a system with rules $r_0 : a \to \{a^{1/2}, \mathtt{T}^{1/2}\}$ and $r_2 : a \to \{a^1\}$.

If we look at Fig. 3, we observe that after two steps, there are *two distinct occurrences* of the element 2, which live in *two different runs* of the program: the run `2.1.2`, and the run `2.3.2`. There are two possible transitions from each 2. The next transition only depends on the fact of having 2, not on the run in which 2 occurs: its history is only a way to distinguish the occurrence. For this reason, given a PARS $(A, \to)$, we keep track of *different occurrences* of an element $a \in A$, but not necessarily of the history. Next section formalizes these ideas.

Markov Decision Processes. To understand our distinction between occurrences of $a \in A$ in different paths, it is helpful to think how a system is described in the framework of Markov Decision Processes (MDP) [33]. Indeed, in the same way as ARS correspond to transition systems, PARS correspond to probabilistic transitions. Let us regard a PARS step $r : a \to \beta$ as a probabilistic transition (r is here a name for the rule). Let assume $a_0 \in \mathcal{A}$ is an initial state. In the setting of MDP, a typical element (called *sample path*) of the sample space Ω is a sequence $\omega = (a_0, r_0, a_1, r_1 \ldots)$ where $r_0 : a_0 \to \beta_1$ is a rule, $a_1 \in \mathtt{Supp}(\beta_1)$ an element, $r_1 : a_1 \to \beta_1$, and so on. The index $t = 0, 1, 2, \dots, n, \dots$ is interpreted as *time*. On Ω various random variables are defined; for example, $X_t = a_t$, which represents the state at time t. The sequence $\langle X_t \rangle$ is called a stochastic process.

3 A Formalism for Probabilistic Rewriting

We introduce a formalism to describe the evolution of a system described by a PARS. From now on, we assume A to be a countable set on which a PARS $(A, \to)$ is defined.

The sample space. Let $\mathtt{m}$ be a list over A, and mA the collection of all such lists. More formally, we fix a countable *index set* $\mathbb{S}$, and let $\mathtt{m} = \{(j, \mathtt{m}_j) \mid j \in \mathbb{S}, \mathtt{m}_j \in A\}$ be the graph of a function from $J \subseteq \mathbb{S}$ to A ($j \mapsto \mathtt{m}_j$). We denote by mA the collection of all such $\mathtt{m}$. $\mathtt{DST}^{\mathtt{F}}(mA) := \bigcup_{\mathtt{m} \in mA} \mathtt{DST}^{\mathtt{F}}(\mathtt{m})$ is the collection of finitely supported distributions μ on $\mathtt{m} \in mA$ (*i.e.* $\mu : \mathtt{m} \to [0, 1]$, with $(j, a) \mapsto p$). For concreteness, here we assume $\mathbb{S} = \mathbb{N}$. Hence, if J is finite, $\mathtt{m}$ is simply a *list* over A.

$\texttt{flat} : (j, a) \mapsto a$

$$(-)^{\texttt{flat}} : \begin{array}{ccc} \texttt{DST}^{\texttt{F}}(mA) & \to & \texttt{DST}^{\texttt{F}}(A) \\ \mu & \mapsto & \mu^{\texttt{flat}} \end{array}$$

where $\mu^{\texttt{flat}}(a) = \mu\{\texttt{flat}^{-1}(a)\} = \sum_{(j,a)\in\texttt{m}} \mu(j, a)$

Figure 6 Flattening.

$\mathcal{I} : a \mapsto (j, a)$

$$(-)^{\mathcal{I}} : \begin{array}{ccc} \texttt{DST}^{\texttt{F}}(A) & \to & \texttt{DST}^{\texttt{F}}(mA) \\ \beta & \mapsto & \beta^{\mathcal{I}} \end{array}$$

where $\beta^{\mathcal{I}}(j, a) = \beta\{\mathcal{I}^{-1}(j, a)\} = \beta(a)$

Figure 7 Embedding.

▶ **Notation 3.1.** *If $\mu \in \texttt{DST}^{\texttt{F}}(mA)$, we write its support as a list. We write $[a, a, b, b]$ for $\{(1, a), (2, a), (3, b), (4, b)\}$ and $[a^{1/4}, a^{1/4}, b^{1/6}, b^{1/3}]$ for $\{(1, a)^{1/4}, (2, a)^{1/4}, (3, b)^{1/6}, (4, b)^{1/3}\}$*

▶ Remark 3.2 (Index Set). The role of indexing is only to distinguish different occurrences; the specific order is irrelevant. We use $\mathbb{N}$ as index set for simplicity. Another natural instance of $\mathbb{S}$ is A^* *i.e.* the set of finite sequences on A. This way, occurrences are labelled by their path, which allows a direct connection with the sample space of Markov Decision Processes [33] we mention in 2.2 (see Appendix).

Given the PARS $\mathcal{A} = (A, \to)$, we work with two families of probability spaces: (A, β), where $\beta \in \texttt{DST}(A)$ (used *e.g.* to describe a rewrite step) and $(\texttt{m}, \mu)$, where $\texttt{m} \in mA$ and $\mu \in \texttt{DST}^{\texttt{F}}(\texttt{m})$.

Letters Convention. we reserve the letters α, β, γ for distributions in $\texttt{DST}(A)$, and the letters $\mu, \nu, \sigma, \tau, \rho, \xi$ for distributions in $\texttt{DST}^{\texttt{F}}(mA)$.

Embedding and Flattening. we move between A and subsets of $\mathbb{N} \times A$ via the maps $\texttt{flat}(-) : \texttt{m} \to A$ and $\mathcal{I} : A \to \texttt{n}$ (Fig. 6 and 7), where to define an injection $\mathcal{I}$, we fix an enumeration $\texttt{n} : \mathbb{N} \to A$, and identify $\texttt{n}$ with its graph. Given a distribution $\mu \in \texttt{DST}^{\texttt{F}}(\texttt{m})$, the function $\texttt{flat}$ induces the distribution $\mu^{\texttt{flat}} \in \texttt{DST}^{\texttt{F}}(A)$ (Fig. 6); conversely, given $\beta \in \texttt{DST}^{\texttt{F}}(A)$, the function $\mathcal{I} : A \to \texttt{n} \in mA$ induces the distributions $\beta^{\mathcal{I}} \in \texttt{DST}^{\texttt{F}}(mA)$ (Fig. 7). Recall that in Sec. 2 we already reviewed how functions induce distributions; indeed, with that language, $\texttt{flat}(-) : \texttt{m} \to A$ and $\mathcal{I} : A \to \texttt{n}$ are random variables.

▶ **Example 3.3.** Assume $\beta = \{a^{0.3}, b^{0.2}, c^{0.5}\}$, and an enumeration of $\{a, b, c\}$. Then $\beta^{\mathcal{I}} = \{(a, 1)^{0.3}, (b, 2)^{0.2}, (c, 3)^{0.5}\}$ which we also write $[a^{0.3}, b^{0.2}, c^{0.5}]$.

Disjoint sum $\uplus$. The disjoint sum of lists is simply their concatenation. The disjoint sum of sets in mA and of the corresponding distributions is easily defined.

The rewriting relation $\rightrightarrows$. Let $\mathcal{A} = (A, \to)$ be a PARS. We now define a binary relation $\rightrightarrows$ on $\texttt{DST}^{\texttt{F}}(mA)$, which is obtained by lifting the relation $\to$. Several natural choices are possible. Here, we choose a lifting which forces *all* non-terminal elements to be reduced. This plays an important role for the development of the paper, as it corresponds to the the key notion of *one step* reduction in classical ARS (see discussion in Sec. 8).

▶ **Definition 3.4** (Lifting). *Given a relation $\to \subseteq A \times \texttt{DST}(A)$, its lifting to a relation $\rightrightarrows \subseteq \texttt{DST}^{\texttt{F}}(mA) \times \texttt{DST}^{\texttt{F}}(mA)$ is defined by the following rules, where for readibility we use Notation 3.1.*

$$\frac{a \in \mathit{NF}_{\mathcal{A}}}{[a^1] \rightrightarrows [a^1]}\ L1 \qquad \frac{a \to \beta \in \mathcal{A}}{[a^1] \rightrightarrows \beta^{\mathcal{I}}}\ L2 \qquad \frac{\left([\texttt{m}_j^1] \rightrightarrows \mu_j\right)_{j\in J}}{[\texttt{m}_j^{p_j} \mid j \in J] \rightrightarrows \biguplus_{j\in J} p_j \cdot \mu_j}\ L3$$

In rule $(L2)$, $\beta^{\mathcal{I}}$ is the result of embedding $\beta \in \mathtt{DST}^{\mathtt{F}}(A)$ in $\mathtt{DST}^{\mathtt{F}}(mA)$ (see Fig. 7 and Example 3.3). To apply rule $(L3)$, we choose a reduction step from $\mathtt{m}_j$ for *each* $j \in J$. The disjoint sum of all μ_j $(j \in J)$ is weighted with the probability of each $\mathtt{m}_j$.

▶ **Example 3.5.** Let us derive the reduction in Fig. 3.

$$\dfrac{2 \to \{1^{1/2}, 3^{1/2}\}}{[2^1] \rightrightarrows [1^{1/2}, 3^{1/2}]} \quad \dfrac{1 \to \{0^{1/2}, 2^{1/2}\} \quad 3 \to \{2^{1/2}, 4^{1/2}\}}{[1^{1/2}, 3^{1/2}] \rightrightarrows [0^{1/4}, 2^{1/4}, 2^{1/4}, 4^{1/4}]} \quad \dfrac{\dots \quad 2 \to \{stop^1\} \quad 2 \to \{1^{1/2}, 3^{1/2}\} \quad \dots}{[0^{1/4}, 2^{1/4}, 2^{1/4}, 4^{1/4}] \rightrightarrows [..., stop^{1/4}, 1^{1/8}, 3^{1/8}, ...]}$$

Rewrite sequences. We write $\mu_0 \rightrightarrows^* \mu_n$ to indicate that there is a *finite sequence* $\mu_0, \dots, \mu_n$ such that $\mu_i \rightrightarrows \mu_{i+1}$ for all $0 \le i < n$ (and $\mu_0 \rightrightarrows^k \mu_k$ to specify its length k). We write $\langle \mu_n \rangle_{n \in \mathbb{N}}$ to indicate an *infinite rewrite sequence.*

Figures conventions. We depict *any* rewrite relation simply as $\to$; as it is standard, we use $\twoheadrightarrow$ for $\to^*$; solid arrows are universally quantified, dashed arrows are existentially quantified.

Normal Forms. The intuition is that a rewrite sequence describes a computation; a distribution μ_i such that $\mu \rightrightarrows^i \mu_i$ represents a state (precisely, the state at time i) in the evolution of the system with initial state μ. Let $\mu \in \mathtt{DST}^{\mathtt{F}}(mA)$ represents a state of the system. The **probability that the system is in normal form** is described by $\mu^{\mathtt{flat}}(\mathtt{NF}_{\mathcal{A}})$ (recall Example 2.1); the probability that the system is in a specific normal form t is described by $\mu^{\mathtt{flat}}(t)$. It is convenient to denote by μ^{NF} the restriction of $\mu^{\mathtt{flat}}$ to $\mathtt{NF}_{\mathcal{A}}$. Observe that $\|\mu^{\mathsf{NF}}\| = \mu^{\mathtt{flat}}(\mathtt{NF}_{\mathcal{A}}) = \mu^{\mathsf{NF}}(\mathtt{NF}_{\mathcal{A}})$. The probability of reaching a normal form t can only increase in a rewrite sequence (becaluse of (L1) in Def. 3.4). Therefore the following key lemma holds.

▶ **Lemma 3.6.** *If* $\sigma \rightrightarrows \tau$ *then* $\sigma^{\mathsf{NF}} \le \tau^{\mathsf{NF}}$.

Equivalences and Order. In this paper we do not need, and do not define, any equality on lists. If we wanted, the natural one would be equality up to reordering, making lists into multisets; however, here we are rather interested in observing specific events. Given $\mu, \rho \in \mathtt{DST}^{\mathtt{F}}(mA)$, we only conside equivalence and order relations w.r.t. the associated (flat) distribution in $\mathtt{DST}(A)$ and in $\mathtt{DST}(\mathtt{NF}_{\mathcal{A}})$. The order on $\mathtt{DST}(A)$ is the pointwise order (Sec. 2).

▶ **Definition 3.7** (Equivalence and Order). *Let* $\mu, \rho \in \mathtt{DST}^{\mathtt{F}}(mA)$.

1. Flat Equivalence*:* $\mathcal{E}_{flat}(\mu, \rho)$*, if* $\mu^{flat} = \rho^{flat}$*. Similarly,* $\le_{flat}$ (μ, ρ) *if* $\mu^{flat} \le \rho^{flat}$*.*
2. Equivalence in Normal Form*:* $\mathcal{E}_{\mathsf{NF}}(\mu, \rho)$*, if* $\mu^{\mathsf{NF}} = \rho^{\mathsf{NF}}$*. Similarly,* $\le_{\mathsf{NF}}$ (μ, ρ)*, if* $\mu^{\mathsf{NF}} \le \rho^{\mathsf{NF}}$
3. Equivalence in the $\mathtt{NF}$-norm*:* $\mathcal{E}_{\|\|_N}(\mu, \rho)$*, if* $\|\mu^{\mathsf{NF}}\| = \|\rho^{\mathsf{NF}}\|$*, and* $\le_{\|\|_N}$ (μ, ρ)*, if* $\|\mu^{\mathsf{NF}}\| \le \|\rho^{\mathsf{NF}}\|$

Observe that (2.) and (3.) compare μ and ρ abstracting from any term which is not in normal form; these two will be the relations which matter to us.

▶ **Example 3.8.** Assume $\mathtt{T}$ is a normal form and $a \neq c$ are not. **(1.)** Let $\mu = [\mathtt{T}^{1/2}, \mathtt{T}^{1/2}]$, $\rho = [\mathtt{T}^1]$. $\mathcal{E}(\mu, \rho)$ holds for $\mathcal{E} \in \{\mathcal{E}_{\mathtt{flat}}, \mathcal{E}_{\mathsf{NF}}, \mathcal{E}_{\|\|_{\mathsf{N}}}\}$ because $\mu^{\mathtt{flat}} = \rho^{\mathtt{flat}} = \{\mathtt{T}^1\}$. **(2.)** Let $\mu = [a^{1/2}, \mathtt{T}^{1/2}]$, $\rho = [c^{1/2}, \mathtt{T}^{1/6}, \mathtt{T}^{2/6}]$. $\mathcal{E}_{\mathsf{NF}}(\mu, \rho), \mathcal{E}_{\|\|_{\mathsf{N}}}(\mu, \rho)$ both *hold*, $\mathcal{E}_{\mathtt{flat}}(\mu, \rho)$ does *not*.

The above example illustrates also the following.

▶ **Fact 3.9.** $\mathcal{E}_{flat}(\mu, \rho) \;\Rightarrow\; \mathcal{E}_{\mathsf{NF}}(\mu, \rho) \;\Rightarrow\; \mathcal{E}_{\|\|_N}(\mu, \rho)$*. Similarly for the order relations.*

4 Asymptotic Behaviour and Normal Forms

We examine the asymptotic behaviour of rewrite sequences *with respect to normal forms.* If a rewrite sequence describes a computation, the *result* of the computation is a distribution on the possible outputs of the probabilistic program. We are interested in the result *at the limit*, which is formalized by the (standard) notion of *limit distribution* (Def. 4.2). What is less standard here, and demands care, is that each termhas a set of limits. In the section we investigate the notions of normalization, termination and unique normal form for PARS.

4.1 Limit Distributions

Before introducing limit distributions, we revisit some facts on sequences of bounded functions.

Monotone Convergence. Let $\langle\alpha_n\rangle_{n\in\mathbb{N}}$ be a *non-decreasing sequence* of (sub)distributions over a countable set X (the order on subdistributions is defined pointwise, Sec. 2). For each $t\in X$, the sequence $\langle\alpha_n(t)\rangle_{n\in\mathbb{N}}$ of real numbers is *nondecreasing and bounded*, therefore the sequence has a limit, which is the supremum: $\lim_{n\to\infty}\alpha_n(t)=\sup_n\{\alpha_n(t)\}$. Observe that if $\alpha<\alpha'$ then $\|\alpha\|<\|\alpha'\|$, where we recall that $\|\alpha\|:=\sum_{x\in X}\alpha(x)$.

▶ **Fact 4.1.** *Given $\langle\alpha_n\rangle_{n\in\mathbb{N}}$ as above, the following properties hold. Define*

$$\beta(t)=\lim_{n\to\infty}\alpha_n(t),\ \forall t\in X$$

1. $\lim_{n\to\infty}\|\alpha_n\| \;=\; \|\beta\|$
2. $\lim_{n\to\infty}\|\alpha_n\|=\sup_n\{\|\alpha_n\|\}\leq 1$
3. β *is a* subdistribution *over* X.

Proof.

1. follows from the fact that $\langle\alpha_n\rangle_{n\in\mathbb{N}}$ is a nondecreasing sequence of functions, hence (by Monotone Convergence, see Thm. A.1 in Appendix) we have :

$$\lim_{n\to\infty}\sum_{t\in X}\alpha_n(t) \;=\; \sum_{t\in X}\lim_{n\to\infty}\alpha_n(t)$$

2. is immediate, because the sequence $\langle\|\alpha_n\|\rangle_{n\in\mathbb{N}}$ is nondecreasing and bounded.
3. follows from (1.) and (2.). Since $\|\beta\|=\sup_n\|\alpha_n\|\leq 1$, then β is a subdistribution. ◀

Limit distributions. Let $\langle\mu_n\rangle_{n\in\mathbb{N}}$ be a rewrite sequence. If $t\in\mathtt{NF}_{\mathcal{A}}$, then $\langle\mu_n^{\mathsf{NF}}(t)\rangle_{n\in\mathbb{N}}$ is nondecreasing (by Lemma 3.6); so we can apply Fact 4.1, with $\langle\alpha_n\rangle_{n\in\mathbb{N}}$ now being $\langle\mu_n^{\mathsf{NF}}\rangle_{n\in\mathbb{N}}$.

▶ **Definition 4.2** (Limits). *Let $\langle\mu_n\rangle_{n\in\mathbb{N}}$ be a rewrite sequence from $\mu\in\mathtt{DST}^{\mathsf{F}}(mA)$. We say*

1. $\langle\mu_n\rangle_{n\in\mathbb{N}}$ ***converges with probability*** $p=\sup_n\|\mu_n^{\mathsf{NF}}\|$.
2. $\langle\mu_n\rangle_{n\in\mathbb{N}}$ ***converges to*** $\beta\in\mathtt{DST}^{\mathsf{F}}(\mathtt{NF}_{\mathcal{A}})$ $\big($*written* $\langle\mu_n\rangle_{n\in\mathbb{N}}\overset{\infty}{\Rrightarrow}\beta\,\big)$, *where for* $t\in\mathtt{NF}_{\mathcal{A}}$

$$\beta(t)=sup_n\{\mu_n^{\mathsf{NF}}(t)\}$$

*We call β a **limit distribution** of μ. We write $\mu\overset{\infty}{\Rrightarrow}\beta$ if μ has a sequence converging to β, and define* $\mathtt{Lim}(\mu):=\{\beta\mid\mu\overset{\infty}{\Rrightarrow}\beta\}$.

4.2 Normalization and Termination

Non-determinism implies that several rewrite sequences are possible from the same $\mu \in \mathtt{DST}^{\mathtt{F}}(mA)$. In the setting of ARS, the notion of reaching a result from a term c comes in two flavours (see Sec. 2): (1.) *there exists* a rewrite sequence from c which leads to a normal form (*normalization*, WN); (2.) *each* rewrite sequence from c leads to a normal form (*termination*, SN). Below, we do a similar $\exists/\forall$ distinction . Instead of reaching a normal form or not, a sequence does so with a probability q.

▶ **Definition 4.3** (Normalization and Termination)**.** *Let $\mu \in \mathtt{DST}^{\mathtt{F}}(mA)$, $q \in [0,1]$. We write $\mu \overset{\infty}{\Rightarrow}_p$ if* there exists a sequence *from μ which converges with probability p.*

- *μ is p-$\mathtt{WN}^{\infty}$ (μ **normalizes** with probability p) if p is the greatest probability to which a sequence from μ can converge.*
- *μ is p-$\mathtt{SN}^{\infty}$ (μ **terminates** with probability p) if* each *sequence from μ converges with probability p. μ is **Almost Sure Terminating (AST)** if it terminates with probability* 1.

A PARS is p-$\mathtt{WN}^{\infty}$, p-$\mathtt{SN}^{\infty}$, AST, if each μ satisfies that property.

▶ **Example 4.4.** The system in Fig. 5 is 1-$\mathtt{WN}^{\infty}$, but not 1-$\mathtt{SN}^{\infty}$. The top rewrite sequence (in blue) converges to $1 = \lim_{n\to\infty} \sum_1^n \frac{1}{2^n}$. The bottom rewrite sequence (in red) converges to 0. In between, we have all dyadic possibilities. In contrast, the system in Fig. 4 is AST.

▶ Remark (Not only AST). Many natural examples are not limited to termination and AST, such as those in Fig. 5, in Example 1.2 and 6.3. For this reason, we go beyond AST, and moreover make a distinction between weak and strong normalization.

4.3 On Unique Normal Forms

How do different rewrite sequences from the same initial μ compare w.r.t. the result they compute? Assume $[M^1] \overset{\infty}{\Rightarrow} \alpha$ and $[M^1] \overset{\infty}{\Rightarrow} \beta$, it is natural to wonder how β and α relate. Normalization and termination are *quantitative yes/no* properties - we are only interested in the measure $\|\beta\|$, for β limit distribution; for example, if $\mu \overset{\infty}{\Rightarrow} \{\mathtt{F}^1\}$ and $\mu \overset{\infty}{\Rightarrow} \{\mathtt{T}^{1/2}, \mathtt{F}^{1/2}\}$, then μ converges with probability 1, but we make no distinction between the two -very different- results. Similarly, consider again Fig. 4. The system is AST, however the limit distributions are *not unique*: they span the continuum $\{\mathtt{T}^p, \mathtt{F}^{1-p}\}$, for $p \in [0,1]$. These observations motivate attention to finer-grained properties.

In Sec. 2 we reviewed the ARS notion of *unique normal form* (UN). Let us now examine an analogue of UN in a probabilistic setting. An intuitive candidate is the following :

$$\mathtt{ULD} : \text{if } \alpha, \beta \in \mathtt{Lim}(\mu), \text{then } \alpha = \beta$$

which was first proposed in [15], where is shown that, in the case of AST, confluence implies ULD. However, ULD is not a good analogue in general, because a PARS does not need to be AST (or $\mathtt{SN}^{\infty}$); it may well be that $\mu \overset{\infty}{\Rightarrow} \alpha$ and $\mu \overset{\infty}{\Rightarrow} \beta$, with $\|\alpha\| \neq \|\beta\|$, as in Ex. 1.2 and in Fig. 5; similar examples are natural in an untyped probabilistic λ-calculus (recall that the λ-calculus is not SN!). In the general case, ULD is not implied by confluence: the system in Fig. 5 is indeed confluent. We then would like to say that it satisfies UN.

We propose as probabilistic analogue of UN the following property

$\mathtt{UN}^{\infty}$: $\mathtt{Lim}(\mu)$ has a *unique maximal* element.

▶ Remark. In the case of $\mathtt{SN}^{\infty}$ (and AST), all limits are maximal, hence $\mathtt{UN}^{\infty}$ becomes ULD.

4.3.1 Confluence and $\mathtt{UN}^\infty$

We justify that $\mathtt{UN}^\infty$ is an appropriate generalization of the $\mathtt{UN}$ property, by showing that it satisfies an analogue of standard ARS results: "Confluence implies $\mathtt{UN}$" (see Thm. 4.7) and "the Normal Form Property implies $\mathtt{UN}$" (Lemma 4.6). While the statements are similar to the classical ones, the content is not. To understand why is different, and non-trivial, observe that $\mathtt{Lim}(\mu)$ is in general uncountable, hence there is not even reason to believe that $\mathtt{Lim}(\mu)$ has maximal elements, for the same reason as $[0,1)$ has no max, even if it has a sup.

▶ Remark 4.5 (Which notion of Confluence?). To guarantee $\mathtt{UN}^\infty$, it suffices a weaker form of confluence than one would expect. Assume $\sigma \ {}^*\!\leftleftarrows \mu \rightrightarrows^* \rho$; with the standard notion of confluence in mind, we may require that $\exists \xi$ such that $\sigma \rightrightarrows^* \xi$, $\rho \rightrightarrows^* \xi$ or that $\exists \xi, \xi'$ such that $\sigma \rightrightarrows^* \xi$, $\rho \rightrightarrows^* \xi'$ and $\mathcal{E}_{\mathtt{flat}}(\xi, \xi')$. Both are fine, but a weaker notion of equivalence suffices: NF-Confluence (defined below), which only regards normal forms. Obviously, the two stronger notions of confluence which we just discussed, imply it.

A PARS satisfies the following properties if they hold for each $\mu \in \mathtt{DST}^{\mathsf{F}}(mA)$:

- **NF-Confluence(Confluence in Normal Form)**:
  $\forall \sigma, \rho$ with $\sigma \ {}^*\!\leftleftarrows \mu \rightrightarrows^* \rho$, $\exists \xi, \tau$ such that $\sigma \rightrightarrows^* \xi$, $\rho \rightrightarrows^* \tau$, and $\xi^{\mathsf{NF}} = \tau^{\mathsf{NF}}$.
- $\mathtt{NFP}^\infty$ **(Normal Form Property):** if α is maximal in $\mathtt{Lim}(\mu)$, and $\mu \rightrightarrows^* \sigma$ then $\sigma \overset{\infty}{\Rrightarrow} \alpha$.
- $\mathtt{LimP}$ **(Limit Distributions Property):** if $\alpha \in \mathtt{Lim}(\mu)$ and $\mu \rightrightarrows^* \sigma$, there exists $\beta \in \mathtt{Lim}(\mu)$ such that $\sigma \overset{\infty}{\Rrightarrow} \beta$ and $\alpha \leq \beta$.

The following result (which is standard for ARS) is easy, and independent from confluence.

▶ **Lemma 4.6.** *For each PARS such that* $\mathtt{Lim}(\mu)$ *has maximal elements,* $\mathtt{NFP}^\infty \Rightarrow \mathtt{UN}^\infty$*.*

Proof. Let $\alpha \in \mathtt{Lim}(\mu)$ be maximal. If $\beta \in \mathtt{Lim}(\mu)$, there is a sequence $\langle \tau_n \rangle_{n \in \mathbb{N}}$ from μ such that $\beta = \sup_n \{\tau_n^{\mathsf{NF}}\}$. $\mathtt{NFP}^\infty$ implies that $\forall n$, $\tau_n \overset{\infty}{\Rrightarrow} \alpha$, and therefore $\tau_n^{\mathsf{NF}} \leq \alpha$. We conclude that $\beta \leq \alpha$; hence if β is maximal, $\beta = \alpha$. ◀

To prove that NF-Confluence implies $\mathtt{UN}^\infty$ is more delicate; the proof is in Appendix. We need to prove that confluence implies existence and uniqueness of maximal elements of $\mathtt{Lim}(\mu)$.

▶ **Theorem 4.7.** *For each PARS, NF-Confluence implies* $\mathtt{UN}^\infty$*.*

Note that the proofs refines those for the analogous ARS properties in a way similar to the generalization to infinitary rewriting, by approximation; the quantitative character of probability add specific elements which are reminiscent of calculus.

4.4 Newman's Lemma Failure, and Proof Technique for PARS

In Prop. 4.6 and 4.7, the statement has the *same flavour* as similar ones for ARS, but the *notions are not the same.* The notion of limit (and therefore that of $\mathtt{UN}^\infty$, $\mathtt{SN}^\infty$, and $\mathtt{WN}^\infty$) does not belong to ARS. For this reason, the rewrite system $(mA, \rightrightarrows)$ which we are studying is not simply an ARS, and one should not assume that standard ARS properties hold. An illustration of this is **Newman's Lemma**. Given a PARS, let us assume $\mathtt{AST}$ and observe that in this case, confluence *at the limit* can be identified with $\mathtt{UN}^\infty$. *A wrong attempt:* $\mathtt{AST}$ + $\mathtt{WCR}^\infty \Rightarrow \mathtt{UN}^\infty$, where $\mathtt{WCR}^\infty$: if $\mu \rightrightarrows \sigma_1$ and $\mu \rightrightarrows \sigma_2$, then $\exists \rho$, with $\sigma_1 \overset{\infty}{\Rrightarrow} \rho$, $\sigma_2 \overset{\infty}{\Rrightarrow} \rho$. This does not hold. A counterexample is the PARS in Fig. 4, which does satisfy $\mathtt{WCR}^\infty$. (More in the Appendix.)

What is at play here is that the notion of *termination* is not the same in ARS and in PARS. A fundamental fact of ARS (on which all proofs of Newman's Lemma rely) is: termination implies that the rewriting relation is well-founded. All terminating ARS allow

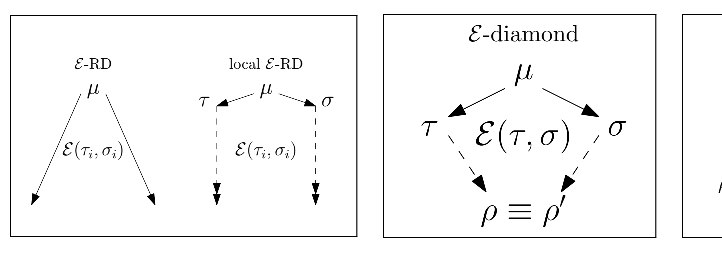


Figure 8 Random Descent. **Figure 9** Diamond. **Figure 10** Proof of 5.4.

well-founded induction as proof technique; this is *not the case* for probabilistic termination. To transfer properties from ARS to PARS there are two issues: we need to find the *right formulation* and the *right proof technique*.

Our counter-example still leaves open the question "Are there *local properties* which guarantee $\mathtt{UN}^\infty$?" In the rest of the paper, we develop proof techniques to study $\mathtt{UN}^\infty$, $\mathtt{WN}^\infty$, $\mathtt{SN}^\infty$ and their relations. We will always aim at *local conditions*.

5 Random Descent (RD)

In this section we introduce $\mathcal{E}$ Random Descent ($\mathcal{E}$-RD), a tool which is able to guarantee some remarkable properties : $\mathtt{UN}^\infty$, *p-termination* as soon as *there exists a sequence* which converges to p, and also the fact that all rewrite sequences from a term have the same *expected* number of steps. $\mathcal{E}$-RD generalizes to PARS the notion of Random Descent: after any k steps, non-determinism is *irrelevant up to a chosen equivalence* $\mathcal{E}$. Indeed $\mathcal{E}$-RD is defined parametrically over an equivalence relation $\mathcal{E}$ on $\mathtt{DST}^{\mathtt{F}}(mA)$. For concreteness, assume $\mathcal{E}$ to be either $\mathcal{E}_{\mathsf{NF}}$ or $\mathcal{E}_{\|\|_{\mathsf{N}}}$ (see Def. 3.7). Then $\mathcal{E}$-RD implies that all rewrite sequences from μ:

- have the same probability of reaching a normal form after k steps (for each $k \in \mathbb{N}$);
- converge to the same limit;
- have the same expected number of steps.

Main technical result is a *local characterization* of the property (Thm 5.4), similarly to [40].

▶ **Definition 5.1** ($\mathcal{E}$ Random Descent). *Let $\mathcal{E}$ be an equivalence relation on $\mathtt{DST}^{\mathtt{F}}(mA)$. The PARS $\mathcal{A}$ satisfies the following properties (in Fig. 8) if they hold for each $\mu \in \mathtt{DST}^{\mathtt{F}}(mA)$.*

- *$\mathcal{E}$-RD: for each pair of sequences $\langle\sigma_n\rangle_{n\in\mathbb{N}}$, $\langle\tau_n\rangle_{n\in\mathbb{N}}$ from μ, $\mathcal{E}(\tau_k, \sigma_k)$ holds, $\forall k$.*
- ***local** $\mathcal{E}$-RD ($\mathcal{E}$-LRD): if $\tau \leftleftarrows \mu \rightrightarrows \sigma$, then for each k there exist σ_k, τ_k with $\sigma \rightrightarrows^k \sigma_k$, $\tau \rightrightarrows^k \tau_k$, and $\mathcal{E}(\sigma_k, \tau_k)$.*

▶ **Example 5.2.** In Fig. 4 $\mathcal{E}$-RD holds for $\mathcal{E} = \mathcal{E}_{\|\|_{\mathsf{N}}}$, but not for $\mathcal{E} = \mathcal{E}_{\mathsf{NF}}$.

When $\mathcal{E} \in \{\mathcal{E}_{\|\|_{\mathsf{N}}}, \mathcal{E}_{\mathsf{NF}}\}$, it is easy to check that $\mathcal{E}$-RD guarantees the following.

▶ **Proposition 5.3.**

1. *$\mathcal{E}_{\|\|_N}$-RD implies **Uniformity**: p-$\mathtt{WN}^\infty \Rightarrow p$-$\mathtt{SN}^\infty$.*
2. *$\mathcal{E}_{NF}$-RD implies Uniformity and $\mathtt{UN}^\infty$.*

Proof. Uniformity is immediate; $\mathtt{UN}^\infty$ follows from Prop. 4.7. ◀

While expressive, $\mathcal{E}$-RD is of little practical use, as it is a property which is *universally quantified* on the sequences from μ. The property $\mathcal{E}$-LRD is instead *local*. Somehow surprisingly, *the local property characterizes* $\mathcal{E}$-RD.

▶ **Theorem 5.4** (Characterization). *The following properties are equivalent:*
1. *$\mathcal{E}$-LRD;*
2. *$\forall k, \mu, \xi, \rho$ if $\mu \rightrightarrows^k \xi$ and $\mu \rightrightarrows^k \rho$, then $\mathcal{E}(\xi, \rho)$;*
3. *$\mathcal{E}$-RD.*

Proof. ($1 \Rightarrow 2$). See Fig. 10. We prove that (2) holds by induction on k. If $k = 0$, the claim is trivial. If $k > 0$, let σ be the first step from μ to ξ and τ the first step from μ to ρ. By $\mathcal{E}$-LRD, there exists σ_k such that $\sigma \rightrightarrows^{k-1} \sigma_k$ and τ_k such that $\tau \rightrightarrows^{k-1} \tau_k$, with $\mathcal{E}(\sigma_k, \tau_k)$. Since $\sigma \rightrightarrows^{k-1} \xi$, we can apply the inductive hypothesis, and conclude that $\mathcal{E}(\sigma_k, \xi)$. By using the induction hypothesis on τ, we have that $\mathcal{E}(\tau_k, \rho)$ and conclude that $\mathcal{E}(\rho, \xi)$. ($2 \Rightarrow 3$). Immediate. ($3 \Rightarrow 1$). Assume $\tau \leftleftarrows \mu \rightrightarrows \sigma$. Take a sequence $\langle \tau_n \rangle_{n \in \mathbb{N}}$ from τ and a sequence $\langle \sigma_n \rangle_{n \in \mathbb{N}}$ from σ. By (3), $\mathcal{E}(\tau_k, \sigma_k)$ $\forall k$. ◀

A diamond. Let $\mathcal{E} \in \{\mathcal{E}_{\mathsf{NF}}, \mathcal{E}_{\|\|_{\mathsf{N}}}\}$. A useful case of $\mathcal{E}$-LRD is the ***$\mathcal{E}$*-diamond** property (Fig. 9): $\forall \mu, \sigma, \tau$, if $\tau \leftleftarrows \mu \rightrightarrows \sigma$, then $\mathcal{E}(\sigma, \tau)$, and $\exists \rho, \rho'$ s.t. $\big(\tau \rightrightarrows \rho, \sigma \rightrightarrows \rho'$ and $\mathcal{E}_{\texttt{flat}}(\rho, \rho')\big)$.

▶ **Proposition 5.5.** *$\mathcal{E}$-diamond $\Rightarrow$ $\mathcal{E}$-LRD.*

Observe that while $\mathcal{E}$-LRD *characterizes* $\mathcal{E}$-RD, $\mathcal{E}$-diamond is only a *sufficient condition*.

5.1 Expected Termination Time

Random Descent captures the property (`Length`) "*all maximal rewrite sequences from a term have the same length*." By looking at ARS as a special case of PARS (with $a \to [b^1]$ for $a \to b$), $\mathcal{E}_{\|\|_{\mathsf{N}}}$-RD does trivialize to RD. More interesting is that $\mathcal{E}_{\|\|_{\mathsf{N}}}$-RD also implies a property similar to (`Length`) for PARS, where we consider not the number of steps of the rewrite sequences, but its probabilistic analogue, the *expected number of steps*.

In an ARS, if a maximal rewrite sequence terminates, the number of steps is finite; we interpret this number as *time to termination*. In the case of PARS, a system may have infinite runs even if it is `AST`; the number of rewrite steps $\to$ from an initial state is (in general) infinite. However, what interests us is its *expected value*, *i.e.* the weighted average w.r.t. probability (see Sec. 2) which we write $\texttt{MeanTime}(\langle \mu_n \rangle_{n \in \mathbb{N}})$. This expected value can be finite; in this case, not only the PARS is `AST`, but is said `PAST` *(Positively* `AST`*)* (see [4]).

▶ **Example 5.6.** In Example 2.5, the sequence from $[a^1]$ has `MeanTime` 2 (see Appendix).

[3] makes a nice observation: the mean number of steps of a rewrite sequence $\langle \mu_n \rangle_{n \in \mathbb{N}}$ admits a very simple formulation, as follows: $\texttt{MeanTime}(\langle \mu_n \rangle_{n \in \mathbb{N}}) = 1 + \sum_{i \geq 1} (1 - \|\mu_i^{\mathsf{NF}}\|)$. Intuitively, each tick in time (*i.e.* each $\rightrightarrows$ step) is weighted with its probability to take place, which is $\mu_i^{\texttt{flat}}\{c \mid c \notin \mathtt{NF}_{\mathcal{A}}\} = 1 - \|\mu_i^{\mathsf{NF}}\|$. Using this formulation, the following result is immediate.

▶ **Corollary 5.7.** *Let $\mu \in \mathtt{DST}^{\mathsf{F}}(mA)$. $\mathcal{E}_{\|\|_N}$-RD implies that all maximal rewrite sequences from μ have the* same `MeanTime`*.*

Observe that $\sum_{i \geq 1} (1 - \|\mu_i^{\mathsf{NF}}\|) < \infty$ implies $\lim_{n \to \infty} (1 - \|\mu_n^{\mathsf{NF}}\|) = 0$, hence $\lim_{n \to \infty} \|\mu_n^{\mathsf{NF}}\| = 1$. Therefore, Cor. 5.7 means that if a sequence from μ with *finite* `MeanTime` exists, μ is `PAST`.

6 Analysis of a probabilistic calculus: *weak* CbV λ-calculus

We introduce $\Lambda_{\oplus}^{\texttt{weak}}$, a probabilistic analogue of call-by-value λ-calculus (see Sec. 1.1). Evaluation is non-deterministic, because in the case of an application there is no fixed order in the evaluation of the left and right subterms (see Example 6.1). We show that $\Lambda_{\oplus}^{\texttt{weak}}$ satisfies

$\mathcal{E}_{\mathsf{NF}}$-RD. Therefore it has remarkable properties (Cor. 6.5), analogous to those of its classical counter-part: the choice of the redex is irrelevant with respect to the *final result*, to its *approximants*, and to the *expected number of steps*.

Syntax. Terms (M, N, P, Q) and values (V, W) are defined as follows:

$$M \quad ::= \quad x \mid \lambda x.M \mid MM \mid M \oplus M \qquad\qquad V \quad ::= \quad x \mid \lambda x.M$$

Free variables are defined as usual. A term M is closed if it has no free variable. The substitution of N for the free occurrences of x in M is denoted $M[x := N]$.

Reductions. Weak call-by-value reduction $\to$ is given as a PARS, and inductively defined by the rules below; its lifting $\rightrightarrows$ is as in Def. 3.4.

$$\begin{array}{l} (\lambda x.M)V \to \{M[x := V]^1\} \\ P \oplus Q \to \{P^{1/2}, Q^{1/2}\} \end{array} \quad \Big| \quad \frac{N \to \{N_i^{p_i} \mid i \in I\}}{MN \to \{MN_i^{p_i} \mid i \in I\}} \qquad \frac{M \to \{M_i^{p_i} \mid i \in I\}}{MN \to \{M_i N^{p_i} \mid i \in I\}}$$

▶ **Example 6.1** (Non-deterministic evaluation). A term may have several redexes. The two reductions here join in one step: $\left[P[x := Q](A \oplus B)^1\right] \leftleftarrows [((\lambda x.P)Q)(A \oplus B)^1] \rightrightarrows [(\lambda x.P)QA^{1/2}, (\lambda x.P)QB^{1/2}]$.

▶ **Example 6.2** (Infinitary reduction). Let $R = (\lambda x.xx \oplus \mathtt{T})(\lambda x.xx \oplus \mathtt{T})$. We have $[R^1] \overset{\infty}{\Rrightarrow} \{\mathtt{T}^1\}$. This term models the behaviour we discussed in Fig.1.

▶ **Example 6.3.** The term PR in Example 1.2 has the following reduction. $[PR^1] \rightrightarrows [P(\mathtt{T} \oplus \mathtt{F})^{1/2}, P(\Delta\Delta)^{1/2}] \rightrightarrows [P(\mathtt{T})^{1/4}, P(\mathtt{F})^{1/4}, P(\Delta\Delta)^{1/2}] \rightrightarrows^* [(\mathtt{T}\ \mathtt{XOR}\ \mathtt{T})^{1/4}, (\mathtt{F}\ \mathtt{XOR}\ \mathtt{F})^{1/4}, \Delta\Delta^{1/2}] \rightrightarrows [\mathtt{F}^{1/4}, \mathtt{F}^{1/4}, \Delta\Delta^{1/2}] \ldots$ We conclude that $PR \overset{\infty}{\Rrightarrow} \{\mathtt{F}^{1/2}\}$.

▶ **Theorem 6.4.** *$\Lambda_\oplus^{weak}$ satisfies $\mathcal{E}$-RD, with $\mathcal{E} = \mathcal{E}_{\mathsf{NF}}$*

Proof. We prove the $\mathcal{E}_{\mathsf{NF}}$-Diamond property, using the definition of lifting and induction on the structure of the terms (see Appendix). ◀

Therefore, by Sec. 5 (and the fact that $\mathcal{E}_{\mathsf{NF}} \Rightarrow \mathcal{E}_{\|\|_{\mathsf{N}}}$), each μ satisfies the following properties:

▶ **Corollary 6.5.**
- All *rewrite sequences from μ converge to the same limit distribution.*
- All *rewrite sequences from μ have the same* expected *termination time* $\mathtt{MeanTime}$.
- *If $\mu \rightrightarrows^k \sigma$ and $\mu \rightrightarrows^k \tau$, then $\sigma^{\mathsf{NF}} = \tau^{\mathsf{NF}}$, $\forall \sigma, \tau, k$.*

More diamonds. Other instances of ARS which satisfy Random Descent are surface reduction in Simpson's linear λ-calculus [38], and Lafont's interaction nets [28]. We do expect that their extension with a probabilistic choice satisfy the same properties as $\Lambda_\oplus^{\mathtt{weak}}$.

7 Comparing Strategies

In this section we provide a method to compare strategies, and a criterion to establish that a strategy is normalizing or perpetual (Cor. 7.4). *When strategy $\mathcal{S}$ is* better *than strategy $\mathcal{T}$?* To study this question we introduce a notion (parametric w.r.t. a relation $\mathcal{R}$) which generalizes the ARS notion of "*better*" introduced in [40]. Like in the case of $\mathcal{E}$-RD, we

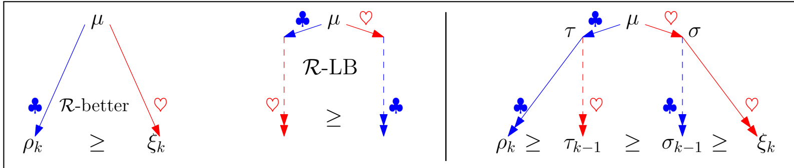


Figure 11 $\mathcal{R}$-better.

will provide a local characterization (Th. 7.3). We obtain criteria which concern both normalization/perpetuity of strategies, and the *expected number of steps* of rewrite sequences.

Given $\mathcal{A} = (A, \to)$, a **rewrite strategy** for $\to$ is a relation $\to_{\mathcal{S}} \subseteq \to$ such that $\mathtt{NF}_{(A,\to_{\mathcal{S}})} = \mathtt{NF}_{\mathcal{A}}$. Let $\rightrightarrows$ (resp. $\rightrightarrows_{\mathcal{S}}$) be the lifting of $\to$ (resp. $\to_{\mathcal{S}}$); we call $\rightrightarrows_{\mathcal{S}}$ a rewrite strategy for $\rightrightarrows$. We indicate by colored arrows $\rightrightarrows_{\clubsuit}$ and $\rightrightarrows_{\heartsuit}$ strategies for $\rightrightarrows$.

▶ **Definition 7.1.** *Given μ, let $p_{\max}(\mu)$ and $p_{\min}(\mu)$ be respectively the* greatest *and* least *value in $\{p \mid \mu \overset{\infty}{\rightrightarrows}_p\}$. A strategy $\rightrightarrows_{\clubsuit}$ is* ***normalizing*** *if for each μ,* each *$\rightrightarrows_{\clubsuit}$-sequence starting from μ converges with probability $p_{\max}(\mu)$. A strategy $\rightrightarrows_{\heartsuit}$ is* ***perpetual*** *if for each μ, each $\rightrightarrows_{\heartsuit}$ sequence from μ converges with probability $p_{\min}(\mu)$.*

Let $\mathcal{R}$ be a *reflexive and transitive relation* on $\mathtt{DST}^{\mathtt{F}}(mA)$. For concreteness, assume $\mathcal{R}$ to be either $\geq_{\|\,\|_{\mathtt{N}}}$ or $\geq_{\mathtt{NF}}$ (see Def. 3.7).

▶ **Definition 7.2** ($\mathcal{R}$-better)**.** *Let $\mathcal{R}$ be a relation as stipulated above. We define the following properties, which are illustrated in Fig. 11.*

- $\rightrightarrows_{\clubsuit}$ ***is $\mathcal{R}$-better*** *than $\rightrightarrows_{\heartsuit}$ ($\mathcal{R}$-better($\rightrightarrows_{\clubsuit}, \rightrightarrows_{\heartsuit}$)): for each μ and for each pair of a $\rightrightarrows_{\clubsuit}$-sequence $\langle \rho_n \rangle_{n\in\mathbb{N}}$ and a $\rightrightarrows_{\heartsuit}$-sequence $\langle \xi_n \rangle_{n\in\mathbb{N}}$ from μ, $\mathcal{R}(\rho_k, \xi_k)$ holds ($\forall k$).*
- $\rightrightarrows_{\clubsuit}$ ***is locally $\mathcal{R}$-better*** *than $\rightrightarrows_{\heartsuit}$ (written $\mathcal{R}$-LB($\rightrightarrows_{\clubsuit}, \rightrightarrows_{\heartsuit}$)): if $\tau \;{}_{\clubsuit}\!\leftleftarrows \mu \rightrightarrows_{\heartsuit} \sigma$, then for each $k \geq 0$, $\exists \sigma_k, \tau_k$, such that $\sigma \rightrightarrows_{\clubsuit}^{k} \sigma_k$, $\tau \rightrightarrows_{\heartsuit}^{k} \tau_k$, and $\mathcal{R}(\tau_k, \sigma_k)$*

By taking $\mathcal{R}$ to be $\geq_{\|\,\|_{\mathtt{N}}}$, it is immediate that $\mathcal{R}$-better($\rightrightarrows_{\clubsuit}, \rightrightarrows$) implies that $\rightrightarrows_{\clubsuit}$ is normalizing. We prove that $\mathcal{R}$-LB is sufficient (and under conditions even necessary) to establish $\mathcal{R}$-better.

▶ **Theorem 7.3.** *Let $\mathcal{R}$ be transitive and reflexive. $\mathcal{R}$-LB($\rightrightarrows_{\clubsuit}, \rightrightarrows_{\heartsuit}$) implies that $\mathcal{R}$-better($\rightrightarrows_{\clubsuit}, \rightrightarrows_{\heartsuit}$). The reverse holds if either $\rightrightarrows_{\clubsuit}$ or $\rightrightarrows_{\heartsuit}$ is $\rightrightarrows$.*

Proof. The proof is illustrated in Fig. 11. The details are in Appendix. ◀

As a consequence, we obtain a method to prove that a strategy is normalizing or perpetual by means of a local condition.

▶ **Corollary 7.4** (Normalizing criterion)**.** *Let $\mathcal{R}(\tau, \sigma)$ be $\geq_{\|\,\|_{\mathtt{N}}} (\tau, \sigma)$ it holds that:*

1. *$\mathcal{R}$-LB($\rightrightarrows_{\clubsuit}, \rightrightarrows$) implies that $\rightrightarrows_{\clubsuit}$ is* normalizing.
2. *$\mathcal{R}$-LB($\rightrightarrows, \rightrightarrows_{\heartsuit}$) implies that $\rightrightarrows_{\heartsuit}$ is* perpetual.

It is easy to check that if $\mathcal{R}$-better($\rightrightarrows_{\clubsuit}, \rightrightarrows$), with $\mathcal{R}$ as above, and $\mathfrak{s}$ is a $\rightrightarrows_{\clubsuit}$-sequence, then $\mathtt{MeanTime}(\mathfrak{s}) \leq \mathtt{MeanTime}(\mathfrak{t})$, for each $\mathfrak{t}$ $\rightrightarrows$-sequence. Therefore, with a similar argument as in Sec.5.1, $\mathcal{R}$-better provides a criterion to establish not only that a strategy is normalizing (resp. perpetual), but also minimality (resp. maximality) of the *expected termination time*.

8 Conclusion and Further Work

We have investigated two properties which are computationally important when studying a calculus whose evaluation is both probabilistic and non-deterministic: *uniqueness of the result* and existence of a *normalizing strategy*. We have defined a probabilistic analogue $\mathtt{UN}^\infty$ of the notion of unique normal form, we have studied conditions which guarantee $\mathtt{UN}^\infty$, and relations with normalization ($\mathtt{WN}^\infty$) and termination ($\mathtt{SN}^\infty$), and between these notions. We have introduced $\mathcal{E}$-RD and $\mathcal{R}$-better as tools to analyze and compare PARS strategies. $\mathcal{E}$-RD is an alternative to strict determinism, analogous to Random Descent for ARS (non-determinism is irrelevant w.r.t. a chosen event of interest). The notion of $\mathcal{R}$-better provides a sufficient criterion to establish that a strategy is *normalizing* (resp. *perpetual*) *i.e.* the strategy is guaranteed to lead to a result with maximal (resp. minimal) probability. We have illustrated the method by studying a probabilistic extension of weak call-by-value λ-calculus; it has analogous properties to its classical counterpart: all rewrite sequences converge to the *same result*, in the same *expected number of steps*.

One-Step Reduction and Expectations. In this paper, we focus on *normal forms* and properties related to the event $\mathtt{NF}_{\mathcal{A}}$. However, we believe that the methods would allow us to compare strategies w.r.t. other properties and random variables of the system. The formalism seems especially well-suited to express the *expected value* of random variables. A key feature of the binary relation $\rightrightarrows$ is to exactly capture the ARS notion of *one-step reduction* (in contrast to *one or no step*), with a gain which is two-folded.

1. *Probability Theory.* Because all terms in the distribution are forced to reduce at the same pace, a rewrite sequence faithfully represents the evolution in time of the system (*i.e.* if $\mu \rightrightarrows^i \mu_i$, then μ_i captures the state at time i of all possible paths $a_0 \to \ldots \to a_i$). This makes the formalism well suited to express the expected value of stochastic processes.
2. *Rewrite Theory.* The results in Sections 5,6,7, crucially rely on *exactly* one-step reduction.

Further work and applications. The motivation behind this work is the need for theoretical tools to support the study of operational properties in probabilistic computation. As an example of application, we mention further work [18] where for each, the Call-by-Value, Call-by-Name, and a linear λ-calculus, a fully fledged probabilistic extension is developed. In each calculus, once establish that given a term, there exists a unique maximal result (the greatest limit distribution), [18] studies the question *"is there a strategy which is guaranteed to reach the unique result (*asymptotic standardization*)?"*. Key elements in [18] rely on the abstract tools developed here; in particular, Sec. 5 and 6 allow us to demonstrate, for both the CbV and CbN *probabilistic* λ-calculi, that the leftmost-outermost strategy reaches the best possible limit distribution. This is remarkable for two reasons. First -as we already observed- the leftmost strategy is the deterministic strategy which is typically adopted in the literature of probabilistic λ-calculus, in either its CbV ([27, 9]) or its CbN version ([13, 16]), but without any completeness result with respect to *probabilistic* computation. [18] offers an "a posteriori" justification for its use. Second, the result is non-trivial, because in the probabilistic case, a standardization result for finite sequences using the leftmost strategy fails for both CbV and CbN. The tools in Sec. 5 allow for an elegant solution.

[40] makes a convincing case of the power of the RD methods for ARS, by using a large range of examples from the literature, to elegantly and uniformly revisit normalization results of various λ-calculi. We cannot here, because the rich development of strategies for λ-calculus has not yet an analogue in the probabilistic case. Nevertheless, we hope that the availability of tools to analyze PARS strategies will contribute to their development.

References

1. Gul A. Agha, José Meseguer, and Koushik Sen. PMaude: Rewrite-based specification language for probabilistic object systems. *Electr. Notes Theor. Comput. Sci.*, 153(2):213–239, 2006.
2. Sheshansh Agrawal, Krishnendu Chatterjee, and Petr Novotný. Lexicographic ranking supermartingales: an efficient approach to termination of probabilistic programs. *PACMPL*, 2(POPL):34:1–34:32, 2018.
3. Martin Avanzini, U. Dal Lago, and Akihisa Yamada. On Probabilistic Term Rewriting. In *Symposium on Functional and Logic Programming, FLOP*, pages 132–148, 2018.
4. Olivier Bournez and Florent Garnier. Proving Positive Almost Sure Termination Under Strategies. In *Rewriting Techniques and Applications, RTA*, pages 357–371, 2006.
5. Olivier Bournez and Claude Kirchner. Probabilistic Rewrite Strategies. Applications to ELAN. In *Rewriting Techniques and Applications, RTA*, pages 252–266, 2002.
6. N. Cagman and J.R. Hindley. Combinatory weak reduction in lambda calculus. *Theor. Comput. Sci.*, 1998.
7. Ugo Dal Lago, Claudia Faggian, Benoît Valiron, and Akira Yoshimizu. The geometry of parallelism: classical, probabilistic, and quantum effects. In *POPL*, pages 833–845, 2017.
8. Ugo Dal Lago and Simone Martini. The weak lambda calculus as a reasonable machine. *Theor. Comput. Sci.*, 398(1-3):32–50, 2008.
9. Ugo Dal Lago, Andrea Masini, and Margherita Zorzi. Confluence Results for a Quantum Lambda Calculus with Measurements. *Electr. Notes Theor. Comput. Sci.*, 270(2):251–261, 2011.
10. Ugo Dal Lago and Margherita Zorzi. Probabilistic operational semantics for the lambda calculus. *RAIRO - Theor. Inf. and Applic.*, 46(3):413–450, 2012.
11. Ugo de'Liguoro and Adolfo Piperno. Non Deterministic Extensions of Untyped Lambda-Calculus. *Inf. Comput.*, 122(2):149–177, 1995.
12. Nachum Dershowitz, Stéphane Kaplan, and David A. Plaisted. Rewrite, Rewrite, Rewrite, Rewrite, Rewrite, . . *Theor. Comput. Sci.*, 83(1):71–96, 1991.
13. Alessandra Di Pierro, Chris Hankin, and Herbert Wiklicky. Probabilistic lambda-calculus and Quantitative Program Analysis. *J. Log. Comput.*, 15(2):159–179, 2005.
14. Alejandro Díaz-Caro, Pablo Arrighi, Manuel Gadella, and Jonathan Grattage. Measurements and Confluence in Quantum Lambda Calculi With Explicit Qubits. *Electr. Notes Theor. Comput. Sci.*, 270(1):59–74, 2011.
15. Alejandro Díaz-Caro and Guido Martinez. Confluence in Probabilistic Rewriting. *Electr. Notes Theor. Comput. Sci.*, 338:115–131, 2018.
16. Thomas Ehrhard, Michele Pagani, and Christine Tasson. The Computational Meaning of Probabilistic Coherence Spaces. In *LICS*, pages 87–96, 2011.
17. Claudia Faggian. Probabilistic Rewriting: On Normalization, Termination, and Unique Normal Forms (Extended Version). Available at `http://arxiv.org/abs/1804.05578`.
18. Claudia Faggian and Simona Ronchi Della Rocca. Lambda Calculus and Probabilistic Computation. In *LICS*, 2019.
19. Luis María Ferrer Fioriti and Holger Hermanns. Probabilistic Termination: Soundness, Completeness, and Compositionality. In *POPL*, pages 489–501, 2015.
20. Hongfei Fu and Krishnendu Chatterjee. Termination of Nondeterministic Probabilistic Programs. In *Verification, Model Checking, and Abstract Interpretation VMCAI*, pages 468–490, 2019.
21. W.A. Howard. Assignment of ordinals to terms for primitive recursive functionals of finite type. In *Intuitionism and Proof Theory*, 1970.
22. Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. Weakest Precondition Reasoning for Expected Runtimes of Randomized Algorithms. *J. ACM*, 65(5):30:1–30:68, 2018.
23. Richard Kennaway. On transfinite abstract reduction systems. Tech. rep., CWI, Amsterdam, 1992.

24 Richard Kennaway, Jan Willem Klop, M. Ronan Sleep, and Fer-Jan de Vries. Transfinite Reductions in Orthogonal Term Rewriting Systems. *Inf. Comput.*, 119(1):18–38, 1995.

25 Maja H. Kirkeby and Henning Christiansen. Confluence and Convergence in Probabilistically Terminating Reduction Systems. In *Logic-Based Program Synthesis and Transformation - 27th International Symposium, LOPSTR 2017*, pages 164–179, 2017.

26 Jan Willem Klop and Roel C. de Vrijer. Infinitary Normalization. In *We Will Show Them! Essays in Honour of Dov Gabbay, Volume Two*, pages 169–192, 2005.

27 Daphne Koller, David A. McAllester, and Avi Pfeffer. Effective Bayesian Inference for Stochastic Programs. In *National Conference on Artificial Intelligence and Innovative Applications of Artificial Intelligence Conference, AAAI 97, IAAI 97*, pages 740–747, 1997.

28 Yves Lafont. Interaction Nets. In *POPL*, pages 95–108, 1990.

29 Simon Marlow. *Parallel and Concurrent Programming in Haskell.* O'Reilly Media, 2013.

30 Annabelle McIver, Carroll Morgan, Benjamin Lucien Kaminski, and Joost-Pieter Katoen. A new proof rule for almost-sure termination. *PACMPL*, 2(POPL):33:1–33:28, 2018.

31 Mark Newman. On Theories with a Combinatorial Definition of "Equivalence". *Annals of Mathematics*, 43(2):223–243, 1942.

32 Sungwoo Park, Frank Pfenning, and Sebastian Thrun. A probabilistic language based upon sampling functions. In *POPL*, pages 171–182, 2005.

33 Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

34 Michael O. Rabin. Probabilistic Automata. *Information and Control*, 6(3):230–245, 1963.

35 Norman Ramsey and Avi Pfeffer. Stochastic lambda calculus and monads of probability distributions. In *POPL*, pages 154–165, 2002.

36 N. Saheb-Djahromi. Probabilistic LCF. In *Mathematical Foundations of Computer Science*, pages 442–451, 1978.

37 Eugene S. Santos. Computability by Probabilistic turing Machines. In *Transactions of the American Mathematical Society*, pages 159:165–184, 1971.

38 Alex K. Simpson. Reduction in a Linear Lambda-Calculus with Applications to Operational Semantics. In *Rewriting Techniques and Applications, RTA*, pages 219–234, 2005.

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

40 Vincent van Oostrom. Random Descent. In *Term Rewriting and Applications, RTA*, page 314–328, 2007.

41 Vincent van Oostrom and Yoshihito Toyama. Normalisation by Random Descent. In *Formal Structures for Computation and Deduction, FSCD*, pages 32:1–32:18, 2016.

A Omitted Proofs and Technical Details

A.1 Appendix to Section 3. A Formalism for Probabilistic Rewriting

The Index Set. A natural choice for the index set $\mathbb{S}$ is $\mathbb{N}$. Another natural choice for the index set $\mathbb{S}$ is A^* *i.e.* the set of finite sequences on A. This way, occurrences of $a \in A$ are labelled by their derivation path. This establishes a direct connection with the sample space of Markov Decision Processes we did mention in Sec. 2.2.

This choice implies that the **embedding** of A in $A^* \times A$ is naturally built-in in the definition of $\rightrightarrows$. Let us rewrite explicitly the definition of lifting. The key point is that the index j in (j, a) records the rewriting path of that occurrence of a. In rule (L2), since we use the rule $a \to \beta$, each $b \in \mathtt{Supp}(\beta)$ is given as index the path $j.a$; $\beta(b)$ is the probability

assigned to b by β. Observe also that all occurrences are automatically distinct.

$$\frac{a \in \mathtt{NF}_{\mathcal{A}}}{\{(j,a)^1\} \rightrightarrows \{(j,a)^1\}}\ L1 \qquad \frac{a \to \beta}{\{(j,a)^1\} \rightrightarrows \{(j.a,b)^{\beta(b)} \mid b \in Supp(\beta)\}}\ L2 \qquad \frac{\left(\{(j,a)^1\} \rightrightarrows \alpha_j\right)_{j \in J}}{\{(j,a)^{p_j} \mid j \in J\} \rightrightarrows \biguplus_{j \in J} p_j \cdot \alpha_j}\ L3$$

A.2 Appendix to Section 4. Asymptotic Behaviour and Normal Forms

Monotone Convergence. We recall the following standard result.

▶ **Theorem A.1** (Monotone Convergence for Sums). *Let X be a countable set, $f_n : X \to [0,\infty]$ a non-decreasing sequence of functions, such that $f(x) := \lim_{n\to\infty} f_n(x) = \sup_n f_n(x)$ exists for each $x \in X$. Then*

$$\lim_{n\to\infty} \sum_{x\in X} f_n(x) \quad = \quad \sum_{x \in \mathcal{X}} f(x)$$

A.2.1 Confluence and $\mathtt{UN}^\infty$: Greatest Limit Distribution

We prove that $\mathtt{NF}$-Confluence implies $\mathtt{UN}^\infty$(Thm. 4.7) *i.e.* confluence implies both *existence and uniqueness of maximal elements* of $\mathtt{Lim}(\mu)$. Both are consequences of $\mathtt{LimP}$ and of the main lemma, Lemma A.2. We recall the definitions.

- $\mathtt{UN}^\infty$: $\mathtt{Lim}(\mu)$ has a *unique maximal* element.
- $\mathtt{NF}$-Confluence: $\forall \sigma, \rho$ with $\sigma \ {}^*\!\leftleftarrows \mu \rightrightarrows^* \rho$, $\exists \xi, \tau$ such that $\sigma \rightrightarrows^* \xi$, $\rho \rightrightarrows^* \tau$, and $\xi^{\mathsf{NF}} = \tau^{\mathsf{NF}}$.
- $\mathtt{LimP}$: if $\alpha \in \mathtt{Lim}(\mu)$ and $\mu \rightrightarrows^* \sigma$, there exists $\beta \in \mathtt{Lim}(\mu)$ such that $\sigma \overset{\infty}{\Rrightarrow} \beta$ and $\alpha \leq \beta$.
- $\mathtt{NFP}^\infty$: if α is maximal in $\mathtt{Lim}(\mu)$, and $\mu \rightrightarrows^* \sigma$ then $\sigma \overset{\infty}{\Rrightarrow} \alpha$.

▶ **Lemma A.2** (Main Lemma). *$\mathtt{NF}$-Confluence implies property* $\mathtt{LimP}$.

Proof. Fig. 12 illustrates the proof. Let $\mu = \rho_0 \in \mathtt{DST}^{\mathsf{F}}(mA)$, and $\langle \rho_n \rangle_{n\in\mathbb{N}}$ be a sequence which converges to α. Assume $\rho_0 \rightrightarrows^* \sigma$. As illustrated in Fig. 12, starting from σ, we build a sequence $\sigma = \sigma_{\rho_0} \rightrightarrows^* \sigma_{\rho_1} \rightrightarrows^* \sigma_{\rho_2} \ldots$, where σ_{ρ_i}, $i \geq 1$ is given by $\mathtt{NF}$-confluence : from $\rho_0 \rightrightarrows^* \sigma_{\rho_{i-1}}$ and $\rho_0 \rightrightarrows^* \rho_i$ we obtain $\sigma_{\rho_{i-1}} \rightrightarrows^* \sigma_{\rho_i}$ and $\rho_i \rightrightarrows^* \tau_i$ with $(\sigma_{\rho_i})^{\mathsf{NF}} = (\tau_i)^{\mathsf{NF}}$. Let β be the limit of the sequence so obtained; observe that $\beta \in \mathtt{Lim}(\rho_0)$. By construction, $\rho_i^{\mathsf{NF}} \leq \tau_i^{\mathsf{NF}} = \sigma_{\rho_i}^{\mathsf{NF}}$; hence $\forall i$, it holds $\rho_i^{\mathsf{NF}} \leq \sigma_{\rho_i}^{\mathsf{NF}} \leq \beta$. From $\alpha = \sup \langle \rho_n^{\mathsf{NF}} \rangle_{n\in\mathbb{N}}$ it follows $\alpha \leq \beta$. ◀

We already established (Lemma 4.6) that $\mathtt{NFP}^\infty$ implies uniqueness of maximal elements (if they exist).

▶ **Corollary A.3** (Uniqueness). $\mathtt{LimP}$ *implies* $\mathtt{NFP}^\infty$.

Proof. Immediate. If α is maximal, then $\beta = \alpha$. ◀

▶ **Lemma A.4** (Existence). $\mathtt{LimP}$ *implies that:*
1. $\mathtt{Norms}(\mu) = \{\|\beta\| \mid \beta \in \mathtt{Lim}(\mu)\}$ *has a greatest element;*
2. $\mathtt{Lim}(\mu)$ *has maximal elements.*

Proof. (1.) Let $p = \sup \mathtt{Norms}(\mu)$. We show that $p \in \mathtt{Norms}(\mu)$, by building a rewrite sequence $\langle \mu_n \rangle_{n\in\mathbb{N}}$ from μ such that $\langle \mu_n \rangle_{n\in\mathbb{N}} \overset{\infty}{\Rrightarrow} \beta$ and $\|\beta\| = p$.

The following facts are all easy to check:

a. If $\alpha < \beta$ then $\|\alpha\| < \|\beta\|$.
b. If $p \notin \mathtt{Norms}(\mu)$, then for each ϵ, there exists $\alpha \in \mathtt{Lim}(\mu)$ such that $\|\alpha\| \geq p - \epsilon$.

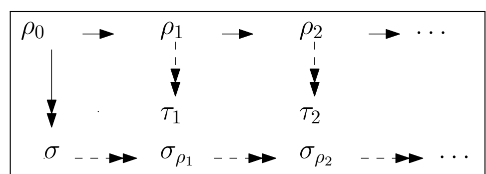


Figure 12 NF-Confluence implies LimP.

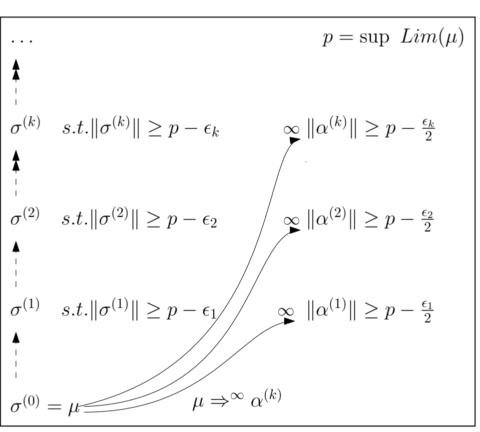


Figure 13 A sequence whose limit distribution is a maximal element of $\texttt{Lim}(\mu)$.

c. LimP implies that, fixed ϵ, if $\mu \overset{\infty}{\Rrightarrow} \alpha$ with $\|\alpha\| \geq (p - \epsilon)$, and $\mu \rightrightarrows^* \sigma$, then there exists σ_{m_ϵ}, such that $\sigma \rightrightarrows^* \sigma_{m_\epsilon}$ and $\|\sigma_{m_\epsilon}\| \geq (p - 2\epsilon)$.
(**Proof**: LimP implies that there is a rewrite sequence $\langle \sigma_n \rangle_{n \in \mathbb{N}}$ from σ which converges to $\gamma \geq \alpha$. Therefore $\langle \sigma_n \rangle_{n \in \mathbb{N}} \overset{\infty}{\Rrightarrow} \gamma$ where $\|\gamma\| \geq (p - \epsilon)$ and $\lim_{n\to\infty} \langle \|\sigma_n\| \rangle = \|\gamma\|$ (by Fact 4.1, point 1.). By definition of limit of a sequence, fixed ϵ, there is an index m_ϵ such that if $m \geq m_\epsilon$ then $\|\sigma_m\| \geq (\|\gamma\| - \epsilon)$, hence $\|\sigma_{m_\epsilon}\| \geq p - 2\epsilon$. Observe that $\sigma \rightrightarrows^* \sigma_{m_\epsilon}$, as a finite rewrite sequence is given by the the first m_ϵ elements of $\langle \sigma_n \rangle_{n \in \mathbb{N}}$.)

d. $\forall \delta \in \mathbb{R}^+$ there exists k such that $\frac{p}{2^k} \leq \delta$.

For each $k \in \mathbb{N}$, let $\epsilon_k = \frac{p}{2^k}$. Let $\sigma^{(0)} = \mu$. From here, we build a sequence of reductions $\mu \rightrightarrows^* \sigma^{(1)} \rightrightarrows^* \sigma^{(2)} \rightrightarrows^* \dots$ whose limit has norm p, as illustrated in Fig. 13.

For each $k > 0$, we observe that:

- By (b.) there exists $\alpha^{(k)} \in \texttt{Lim}(\mu)$ such that $\|\alpha^{(k)}\| \geq (p - \frac{1}{2}\frac{p}{2^k})$.
- From $\mu \rightrightarrows^* \sigma^{(k-1)}$, we use (c.) to establish that there exists $\sigma^{(k)}$ such that $\sigma^{(k-1)} \rightrightarrows^* \sigma^{(k)}$ and $\|\sigma^{(k)}\| \geq (p - \frac{p}{2^k})$. Observe that $\alpha^{(k)}$, $\sigma^{(k-1)}$, $\sigma^{(k)}$ resp. instantiate α, σ, σ_{m_ϵ} of (c.).

Let $\langle \mu_n \rangle_{n \in \mathbb{N}}$ be the concatenation of all the finite sequences $\sigma^{(k-1)} \rightrightarrows^* \sigma^{(k)}$, and let β be its limit distribution. By construction, $\lim_{n\to\infty} \langle \|\mu_n\| \rangle = \|\beta\| = p$, hence $p \in \texttt{Norms}(\mu)$.

(1. $\Rightarrow$ 2.) We observe that if $\langle \mu_n \rangle_{n \in \mathbb{N}} \overset{\infty}{\Rrightarrow} \alpha$ and $\|\alpha\|$ is maximal in $\texttt{Norms}(\mu)$, then α is maximal in $\texttt{Lim}(\mu)$ (because if $\gamma \in \texttt{Lim}(\mu)$ and $\gamma > \alpha$, then $\|\gamma\| > \|\alpha\|$). ◀

▶ **Theorem 4.7** (restated). *For each PARS, NF-Confluence implies* $\texttt{UN}^\infty$.

Proof. The claim follows from the Main Lemma, A.3 and A.4 (point 2.), using Lemma 4.6. ◀

Semi-Confluence. The proof of Thm. 4.7 shows we can weaken confluence even further.

▶ **Proposition A.5.** *For each PARS, the property below implies* LimP.
Semi-Confluence*:* $\forall \mu, \sigma, \rho$ *with* $\sigma \ {}^*\!\leftleftarrows \mu \rightrightarrows^* \rho$, $\exists \sigma'$ *such that* $\sigma \rightrightarrows^* \sigma'$ *and* $\rho^{NF} \leq \sigma'^{NF}$.

Proof. Proof A.2 really only uses Semi-Confluence. Let us revisit the proof, which is now illustrated in Fig. 14. Let $\rho_0 \in \texttt{DST}^\texttt{F}(mA)$, and α maximal in $\texttt{Lim}(\rho_0)$. Assume $\rho_0 \rightrightarrows^* \sigma$. Let $\langle \rho_n \rangle_{n \in \mathbb{N}}$ be a sequence which converges to α. As illustrated in Fig. 14, starting from σ, we build a sequence $\sigma = \sigma_{\rho_0} \rightrightarrows^* \sigma_{\rho_1} \rightrightarrows^* \sigma_{\rho_2} \dots$, where σ_{ρ_i}, $i \geq 1$ is given by Semi-Confluence:

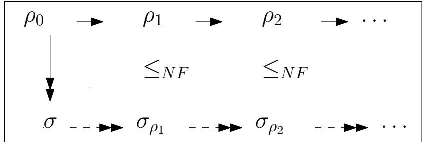


Figure 14 Semi-Confluence implies LimP.

from $\rho_0 \rightrightarrows^* \rho_i$ and $\rho_0 \rightrightarrows^* \sigma_{\rho_{i-1}}$ we obtain $\sigma_{\rho_{i-1}} \rightrightarrows^* \sigma_{\rho_i}$ with $\rho_i^{\mathsf{NF}} \leq \sigma_{\rho_i}^{\mathsf{NF}}$. Let β be the limit of the sequence so obtained; observe that $\beta \in \mathtt{Lim}(\rho_0)$. By construction, $\forall i$, it holds $\rho_i^{\mathsf{NF}} \leq (\sigma_{\rho_i})^{\mathsf{NF}} \leq \beta$. From $\alpha = \sup \langle \rho_n^{\mathsf{NF}} \rangle_{n\in\mathbb{N}}$ it follows that $\alpha \leq \beta$. ◀

A.2.2 More on Newman's Lemma and Proof Techniques

We pointed out that to transfer properties from ARS to PARS there are two issues: find the *right formulation* and the *right proof technique*. Newman's Lemma illustrates both. Can a different formulation uncover properties similar to Newman Lamma? Another "candidate" statement we can attempt is : $\mathtt{AST}$ + WCR $\Rightarrow \mathtt{UN}^\infty$. I have no answer here. This property is indeed an interesting case study. It is not hard to show that this property holds when $\mathtt{Lim}(\mu)$ is finite, or uniformly discrete, which also means that a counterexample (if any) cannot be trivial. On the other side, if the property holds, the difficulty is which proof technique to use, since well-founded induction is not available to us.

A.3 Appendix to Section 5. Random Descent

$\mathcal{E}$-diamond implies $\mathcal{E}$-LRD. In this section, we assume $\mathcal{E} \in \{\mathcal{E}_{\mathsf{NF}}, \mathcal{E}_{\|\|_{\mathsf{N}}}\}$.

▶ **Lemma A.6.** *If $\mathcal{E}_{flat}(\mu, \rho,)$, there exists a rewrite sequence $\langle \mu_n \rangle_{n\in\mathbb{N}}$ and a rewrite sequence $\langle \rho_n \rangle_{n\in\mathbb{N}}$, with $\mathcal{E}_{flat}(\mu_i, \rho_i,)$*

Proof. By easy induction on n. It is enough, at each $\rightrightarrows$ step as defined in Def. 3.4, to choose the same reduction $c \to \beta$ for all $(j, \mathtt{m}_j)$ such that $\mathtt{m}_j = c$. ◀

▶ **Proposition A.7.** *$\mathcal{E}$-diamond $\Rightarrow$ $\mathcal{E}$-LRD.*

Proof. By using Lemma A.6. ◀

Point-wise formulation. In Section 6, we exploit the fact that not only $\mathcal{E}$-RD admits a local characterization, but also that the properties $\mathcal{E}$-LRD and $\mathcal{E}$-diamond can be expressed point-wise, making the condition easier to verify.

1. pointed $\mathcal{E}$-LRD: $\forall a \in A$, if $\tau \leftleftarrows [a^1] \rightrightarrows \sigma$, then $\forall k, \exists \sigma_k, \tau_k$ with $\sigma \rightrightarrows^k \sigma_k$, $\tau \rightrightarrows^k \tau_k$, and $\mathcal{E}(\sigma_k, \tau_k)$.
2. pointed $\mathcal{E}$-diamond: $\forall a \in A$, if $\tau \leftleftarrows [a^1] \rightrightarrows \sigma$, then it holds that $\mathcal{E}(\sigma, \tau)$, and $\exists \rho, \rho'$ such that $\tau \rightrightarrows \rho, \sigma \rightrightarrows \rho'$ and $\mathcal{E}_{\mathtt{flat}}(\rho, \rho',)$.

▶ **Proposition A.8** (point-wise $\mathcal{E}$-LRD). *The following hold*

- *$\mathcal{E}$-LRD $\iff$ pointed $\mathcal{E}$-LRD;*
- *$\mathcal{E}$-diamond $\iff$ pointed $\mathcal{E}$-diamond.*

Proof. Immediate, by the definition of $\rightrightarrows$. Given $\mu = [a_i^{p_i} \mid i \in I]$, we establish the result for each a_i, and put all the resulting distributions together. ◀

A.3.1 Section 5.1. Finite expected time to termination

An example of PARS with finite expected time to termination is the one in Fig. 1. We can see this informally, recalling Sec. 2. Let the sample space Ω be the set of paths ending in a normal form, and let μ be the probability distribution on Ω. What is the expected value of the random variable $\mathtt{length} : \Omega \to \mathbb{N}$? We have $E(\mathtt{length}) = \sum_\omega \mathtt{length}(\omega) \cdot \mu(\omega) = \sum_{n\in\mathbb{N}} n \cdot \mu\{\omega \mid \mathtt{length}(\omega) = n\} = \sum n \cdot \frac{1}{2^n} = 2$.

It is immediate to check that in Example 2.5, the (only) rewrite sequence from $[a^1]$ has MeanTime 2 by using the definition of mean number of steps of a rewrite sequence $\langle\mu_n\rangle_{n\in\mathbb{N}}$ as

$$\mathtt{MeanTime}(\langle\mu_n\rangle_{n\in\mathbb{N}}) = 1 + \sum_{i\geq 1}(1 - \|\mu_i^{\mathsf{NF}}\|)$$

as formulated in [3] (to which we refer for the details).

A.4 Appendix to Section 6. Weak CbV λ-calculus

We prove Thm. 6.4.

▶ **Theorem 6.4** (restated). $\Lambda_\oplus^{weak}$ *satisfies the* $\mathcal{E}$*-diamond property, with* $\mathcal{E} = \mathcal{E}_{\mathsf{NF}}$.

Proof. We show by induction on the structure of the term M that for all pairs of one-step reductions $\tau \leftleftarrows [M^1] \rightrightarrows \sigma$, the following hold: (1.) $\sigma^{\mathsf{NF}} = \tau^{\mathsf{NF}}$ is 0 everywhere (2.) exists ρ, ρ' such that $\tau \rightrightarrows \rho$, $\sigma \rightrightarrows \rho'$ and $\mathcal{E}_{\mathtt{flat}}(\rho, \rho',)$.

It is convenient to introduce the following notation. If $\mu = [M_i^{m_i} \mid i \in I]$ we define

$$\mu @ Q := [(M_i Q)^{m_i} \mid i \in I] \quad Q @ \mu := [(Q M_i)^{m_i} \mid i \in I]$$

We write the Dirac distribution $[\mathtt{m}_i^1]$ simply as $[\mathtt{m}_i]$. We write $\rho \equiv \rho'$ for $\mathcal{E}_{\mathtt{flat}}(\rho, \rho',)$.

- If $M = x$ or $M = \lambda x.P$, no reduction is possible.
- If $M = P \oplus_p Q$, only one reduction is possible.
- If $M = PQ$ is a redex, then $P = (\lambda x.N)$, Q is a value, and no other reduction is possible inside either P or Q.
- If $M = PQ$ has two different reductions, two cases are possible.
  - Assume that both P and Q reduce; PQ has the following reductions.

    $$\frac{P \to \{P_i^{p_i} \mid i \in I\}}{PQ \to \sigma = \{P_i Q^{p_i} \mid i \in I\}} \quad \text{and} \quad \frac{Q \to \{Q_j^{q_j} \mid j \in J\}}{PQ \to \tau = \{PQ_j^{q_j} \mid j \in J\}}$$

    Observe that none of the P_iQ or PQ_j is a normal form, hence (1.) holds. By the definition of reduction, the following holds

    $$\frac{Q \to \{Q_j^{q_j} \mid j \in J\}}{P_i Q \to \{P_i Q_j^{q_j} \mid j \in J\}}$$

    and therefore by Lifting we have $\biguplus_i p_i \cdot [P_i Q] \rightrightarrows \biguplus_i p_i \cdot (\biguplus_j q_j \cdot [P_i Q_j]) = \biguplus_{i,j} p_i q_j \cdot [P_i Q_j]$. Similarly we obtain $\biguplus_j q_j \cdot [PQ_j] \rightrightarrows \biguplus_{i,j} p_i q_j \cdot [P_i Q_j]$.
  - Assume that one subterm has two different redexes; let assume it is the subterm P (the case of Q is similar):

    $$P \to \sigma = \{S_i^{s_i} \mid i \in I\} \text{and} P \to \tau = \{T_j^{t_j} \mid j \in J\}$$

By inductive hypothesis, two facts hold: (1.) $\sigma^{\mathsf{NF}} = \tau^{\mathsf{NF}}$ is 0 everywhere, therefore no S_i and no T_j in the support is a normal form; (2.) exists ρ, ρ' with $\rho \equiv \rho'$ and such that $[S_i] \rightrightarrows \rho_i$ with $\sum_i s_i \cdot \rho_i = \rho$, and $[T_j] \rightrightarrows \rho_j$ with $\sum_j t_j \cdot \rho_j = \rho'$. For PQ we have

$$\frac{P \to \{S_i^{s_i} \mid i \in I\}}{PQ \to \{(S_iQ)^{s_i} \mid i \in I\}} \quad \text{and} \quad \frac{P \to \{T_j^{t_j} \mid j \in J\}}{PQ \to \{(T_jQ)^{t_j} \mid j \in J\}}$$

First of all, we observe that no S_iQ and no T_jQ is a normal form, hence property (1.) is verified. Moreover, it holds that $S_iQ \to \rho_i@Q$ and $T_jQ \to \rho_j@Q$. We conclude by Lifting that $[(S_iQ)^{s_i} \mid i \in I] \rightrightarrows \biguplus_i s_i \cdot \rho_i@Q$. It is easy to check that, $\biguplus_i s_i \cdot \rho_i@Q = \rho@Q$, and $[(T_jQ)^{t_j} \mid j \in J] \rightrightarrows \biguplus t_j \cdot \rho_j@Q = \rho@Q$. It is immediate also that $\rho@Q \equiv \rho'@Q$; hence property (2.) is also verified. ◀

A.5 Appendix to Section 7. Comparing Strategies

▶ **Theorem 7.3** (restated). *Let $\mathcal{R}$ be transitive and reflexive. $\mathcal{R}$-LB($\rightrightarrows_{\clubsuit}, \rightrightarrows_{\heartsuit}$) implies that $\mathcal{R}$-better($\rightrightarrows_{\clubsuit}, \rightrightarrows_{\heartsuit}$). The reverse holds if either $\rightrightarrows_{\clubsuit}$ or $\rightrightarrows_{\heartsuit}$ is $\rightrightarrows$.*

Proof. ⇒ . See Fig. 11. We prove by induction on k the following: "*$\mathcal{R}$-LB($\rightrightarrows_{\clubsuit}, \rightrightarrows_{\heartsuit}$) implies ($\forall \mu, \rho, \xi$, if $\mu \rightrightarrows_{\clubsuit}{}^k \rho$ and $\mu \rightrightarrows_{\heartsuit}{}^k \xi$, then $\mathcal{R}(\rho, \xi)$)*". If $k = 0$, the claim is trivial. If $k \geq 1$, let σ be the first step from μ to ξ, and τ the first step from μ to ρ, as in Fig. 11. $\mathcal{R}$-LB implies that exist σ_{k-1} and τ_{k-1} such that $\sigma \rightrightarrows_{\clubsuit}{}^{k-1} \sigma_{k-1}$, $\tau \rightrightarrows_{\heartsuit}{}^{k-1} \tau_{k-1}$, with $\mathcal{R}(\tau_{k-1}, \sigma_{k-1})$. Since $\sigma \rightrightarrows_{\heartsuit}{}^{k-1} \xi$ we can apply the inductive hypothesis, and obtain that $\mathcal{R}(\sigma_{k-1}, \xi)$. Again by inductive hypothesis, from $\tau \rightrightarrows_{\clubsuit}{}^{k-1} \rho$ we obtain $\mathcal{R}(\rho, \tau_{k-1})$. By transitivity, it holds that $\mathcal{R}(\rho, \xi)$. ⇐ . Assume $\rightrightarrows_{\heartsuit} = \rightrightarrows$, and $\tau \;{}_{\clubsuit}\!\leftleftarrows \mu \rightrightarrows \sigma$. Let $\langle \tau_n \rangle_{n \in \mathbb{N}}$ and $\langle \sigma_n \rangle_{n \in \mathbb{N}}$ be obtained by extending τ and σ with a maximal $\rightrightarrows_{\clubsuit}$ sequence. The claim follows from the hypothesis that $\rightrightarrows_{\clubsuit}$ dominates $\rightrightarrows$, by viewing the $\rightrightarrows_{\clubsuit}$ steps in $\langle \sigma_n \rangle_{n \in \mathbb{N}}$ as $\rightrightarrows$ steps. ◀

▶ Remark. Observe that $\mathcal{E}$-RD (resp. $\mathcal{E}$-LRD) is a special case of $\mathcal{R}$-better (resp. $\mathcal{R}$-LB). We have preferred to treat it independently, for the sake of presentation.

A.6 Comments

Finite Approximants. $\mathcal{E}$-RD characterizes the case when (not only at the limit, but also at the level of the approximants) the non-deterministic choices are irrelevant. The notion of approximant which we have studied here is "stop after a number k of steps" ($k \in \mathbb{N}$). We can consider different notion of approximants. For example, we could also wish to stop the evolution of the system when it reaches a normal form with probability p. Our method can easily be adapted to analyze this case (see [17]). We believe it is also possible to extend to the probabilistic setting the results in [41], which would allow to further push this direction.

The beauty of local. In Sec. 5 and 7, by local conditions we mean the following: to show that a property P holds globally (*i.e. for each* two rewrite sequences, P holds), we can show that P holds locally (*i.e.* for each pair of one-step reductions, *there exist* two rewrite sequences such that P holds). This reduces the space of search for testing the property, a fact that we exploit in the proofs of Sec. 6.

A Linear-Logical Reconstruction of Intuitionistic Modal Logic S4

Yosuke Fukuda
Graduate School of Informatics, Kyoto University, Japan
http://www.fos.kuis.kyoto-u.ac.jp/~yfukuda
yfukuda@fos.kuis.kyoto-u.ac.jp

Akira Yoshimizu
French Institute for Research in Computer Science and Automation (INRIA), France
http://www.cs.unibo.it/~akira.yoshimizu
akira.yoshimizu@inria.fr

Abstract

We propose a *modal linear logic* to reformulate intuitionistic modal logic S4 (IS4) in terms of linear logic, establishing an S4-version of Girard translation from IS4 to it. While the Girard translation from intuitionistic logic to linear logic is well-known, its extension to modal logic is non-trivial since a naive combination of the S4 modality and the exponential modality causes an undesirable interaction between the two modalities. To solve the problem, we introduce an extension of intuitionistic multiplicative exponential linear logic with a modality combining the S4 modality and the exponential modality, and show that it admits a sound translation from IS4. Through the Curry–Howard correspondence we further obtain a Geometry of Interaction Machine semantics of the modal λ-calculus by Pfenning and Davies for staged computation.

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

Keywords and phrases linear logic, modal logic, Girard translation, Curry–Howard correspondence, geometry of interaction, staged computation

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.20

Acknowledgements The authors thank Kazushige Terui for his helpful comments on our work and pointing out related work on multicolored linear logic, and Atsushi Igarashi for his helpful comments on earlier drafts. Special thanks are also due to anonymous reviewers for their fruitful comments. This work was supported by the Research Institute for Mathematical Sciences, an International Joint Usage/Research Center located in Kyoto University.

1 Introduction

Linear logic discovered by Girard [7] is, as he wrote, not an alternative logic but should be regarded as an "extension" of usual logics. Whereas usual logics such as classical logic and intuitionistic logic admit the structural rules of weakening and contraction, linear logic does not allow to use the rules freely, but it reintroduces them in a controlled manner by using the exponential modality '!' (and its dual '?'). Usual logics are then reconstructed in terms of linear logic with the power of the exponential modalities, via the Girard translation.

In this paper, we aim to extend the framework of linear-logical reconstruction to the $(\Box, \supset)$-fragment of intuitionistic modal logic S4 (IS4) by establishing what we call "modal linear logic" and an S4-version of Girard translation from IS4 into it. However, the crux to give a faithful translation is that a naive combination of the $\Box$-modality and the !-modality causes an undesirable interaction between the inference rules of the two modalities. To solve

© Yosuke Fukuda and Akira Yoshimizu;
licensed under Creative Commons License CC-BY
4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019).
Editor: Herman Geuvers; Article No. 20; pp. 20:1–20:24
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Syntactic category	Inference rule
Formulae $A, B, C ::= p \mid A \multimap B \mid {!A}$	$\dfrac{}{A \vdash A}\,\mathrm{Ax}$ $\quad \dfrac{\Gamma \vdash A \quad \Gamma', A \vdash B}{\Gamma, \Gamma' \vdash B}\,\mathrm{Cut}$ $\quad \dfrac{!\Gamma \vdash A}{!\Gamma \vdash !A}\,!\mathrm{R}$

$$\dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B}\multimap\mathrm{R} \quad \dfrac{\Gamma \vdash A \quad \Gamma', B \vdash C}{\Gamma, \Gamma', A \multimap B \vdash C}\multimap\mathrm{L} \quad \dfrac{\Gamma, A \vdash B}{\Gamma, !A \vdash B}\,!\mathrm{L} \quad \dfrac{\Gamma \vdash B}{\Gamma, !A \vdash B}\,!\mathrm{W} \quad \dfrac{\Gamma, !A, !A \vdash B}{\Gamma, !A \vdash B}\,!\mathrm{C}$$

Figure 1 Definition of IMELL.

$$\underline{\lceil A \rceil} \quad \lceil p \rceil \stackrel{\text{def}}{=} p, \qquad \lceil A \supset B \rceil \stackrel{\text{def}}{=} (!\lceil A \rceil) \multimap \lceil B \rceil \qquad \underline{\lceil \Gamma \rceil} \quad \lceil \Gamma \rceil \stackrel{\text{def}}{=} \{\lceil A \rceil \mid A \in \Gamma\}$$

Figure 2 Definition of the Girard translation from intuitionistic logic.

the problem, we define the modal linear logic as an extension of intuitionistic multiplicative exponential linear logic with a modality '$\boxdot$' (pronounced by "bangbox") that integrates '$\Box$' and '!', and show that it admits a faithful translation from IS4.

As an application, we consider a computational interpretation of the modal linear logic. A typed λ-calculus that we will define corresponds to a natural deduction for the modal linear logic through the Curry–Howard correspondence, and it can be seen as a reconstruction of the modal λ-calculus by Pfenning and Davies [18, 5] for the so-called staged computation. Thanks to our linear-logical reconstruction, we can further obtain a Geometry of Interaction Machine (GoIM) for the modal λ-calculus.

The remainder of this paper is organized as follows. In Section 2 we review some formalizations of linear logic and IS4. In Section 3 we explain a linear-logical reconstruction of IS4. First, we discuss how a naive combination of linear logic and modal logic fails to obtain a faithful translation. Then, we propose a modal linear logic with the $\boxdot$-modality that admits a faithful translation from IS4. In Section 4 we give a computational interpretation of modal linear logic through a typed λ-calculus. In Section 5 we provide an axiomatization of modal linear logic by a Hilbert-style deductive system. In Section 6 we obtain a GoIM of our typed λ-calculus as an application of our linear-logical reconstruction. In Sections 7 and 8 we discuss related work and conclude our work, respectively.

2 Preliminaries

We recall several systems of linear logic and modal logic. In this paper, we consider the minimal setting to give an S4-version of Girard translation and its computational interpretation. Thus, every system we will use only contain an implication and a modality as operators.

2.1 Intuitionistic MELL and its Girard translation

Figure 1 shows the standard definition of the $(!, \multimap)$-fragment of *intuitionistic multiplicative exponential linear logic*, which we refer to as IMELL. A formula is either a propositional variable, a linear implication, or an exponential modality. We let p range over the set of propositional variables, and A, B, C range over formulae. A *context* Γ is defined to be a multiset of formulae, and hence the exchange rule is assumed as a meta-level operation. A *judgment* consists of a context and a formula, written as $\Gamma \vdash A$. As a convention, we often write $\Gamma \vdash A$ to mean that the judgment is derivable (and we assume similar conventions throughout this paper). The notation $!\Gamma$ in the rule !R denotes the multiset $\{!A \mid A \in \Gamma\}$.

Syntactic category

Formulae $A, B, C ::= p \mid A \supset B \mid \Box A$

Inference rule

$$\frac{}{A \vdash A}\,\mathrm{Ax} \qquad \frac{\Gamma \vdash A \quad \Gamma', A \vdash B}{\Gamma, \Gamma' \vdash B}\,\mathrm{Cut} \qquad \frac{\Box\Gamma \vdash A}{\Box\Gamma \vdash \Box A}\,\Box\mathrm{R}$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \supset B}\,\supset\!\mathrm{R} \qquad \frac{\Gamma \vdash A \quad \Gamma', B \vdash C}{\Gamma, \Gamma', A \supset B \vdash C}\,\supset\!\mathrm{L} \qquad \frac{\Gamma, A \vdash B}{\Gamma, \Box A \vdash B}\,\Box\mathrm{L} \qquad \frac{\Gamma \vdash B}{\Gamma, A \vdash B}\,\mathrm{W} \qquad \frac{\Gamma, A, A \vdash B}{\Gamma, A \vdash B}\,\mathrm{C}$$

Figure 3 Definition of $\mathrm{LJ}^{\Box}$.

Figure 2 defines the Girard translation[1] from the ⊃-fragment of intuitionistic propositional logic IL. For an IL-formula A, $\lceil A \rceil$ will be an IMELL-formula; and $\lceil \Gamma \rceil$ a multiset of IMELL-formulae. Then, we can show that the Girard translation from IL to IMELL is sound.

▶ **Theorem 1** (Soudness of the translation). *If* $\Gamma \vdash A$ *in* IL*, then* $!\lceil \Gamma \rceil \vdash \lceil A \rceil$ *in* IMELL*.*

2.2 Intuitionistic S4

We review a formalization of the $(\Box, \supset)$-fragment of intuitionistic propositional modal logic S4 (IS4). In what follows, we use a sequent calculus for the logic, called $\mathrm{LJ}^{\Box}$. The calculus $\mathrm{LJ}^{\Box}$ used here is defined in a standard manner in the literature (e.g. it can be seen as the IS4-fragment of **G1s** for classical modal logic S4 by Troelstra and Schwichtenberg [23]).

Figure 3 shows the definition of $\mathrm{LJ}^{\Box}$. A formula is either a propositional variable, an intuitionistic implication, or a box modality. A *context* and a *judgment* are defined similarly in IMELL. The notation $\Box\Gamma$ in the rule $\Box$R denotes the multiset $\{\Box A \mid A \in \Gamma\}$.

▶ Remark 2. It is worth noting that the !-exponential in IMELL and the $\Box$-modality in $\mathrm{LJ}^{\Box}$ have similar structures. To see this, let us imagine the rules $\Box$R and $\Box$L replacing the symbol '$\Box$' with '!'. The results will be exactly the same as !R and !L. In fact, the !-exponential satisfies the S4 axiomata in IMELL, which is the reason we also call it as a modality.

2.3 Typed λ-calculus of the intuitionistic S4

We review the modal λ-calculus developed by Pfenning and Davies [18, 5], which we call $\lambda^{\Box}$. The system $\lambda^{\Box}$ is essentially the same calculus as $\lambda_{e}^{\to\Box}$ in [5], although some syntax are changed to fit our notation in this paper. $\lambda^{\Box}$ is known to correspond to a natural deduction system for IS4, as is shown in [18].

Figure 4 shows the definition of $\lambda^{\Box}$. The set of types corresponds to that of formulae of IS4. We let x range over the set of term variables, and M, N, L range over the set of terms. The first three terms are as in the simply-typed λ-calculus. The terms $\Box M$ and $\textbf{let}\,\Box x = M\,\textbf{in}\,N$ is used to represent a constructor and a destructor for types $\Box A$, respectively. The variable x in $\lambda x : A.M$ and $\textbf{let}\,\Box x = M\,\textbf{in}\,N$ is supposed to be *bound* in the usual sense and the *scope* of the biding is M and N, respectively. The set of *free* (i.e., unbound) variables in M is denoted by $\mathrm{FV}(M)$. We write the *capture-avoiding substitution* $M[x := N]$ to denote the result of replacing N for every free occurrence of x in M.

A *(type) context* is defined to be the set of pairs of a term variable x_i and a type A_i such that all the variables are distinct, which is written as $x_1 : A_1, \cdots, x_n : A_n$ and is denoted by Γ, Δ, Σ, etc. Then, a *(type) judgment* is defined, in the so-called *dual-context* style, to consists of two contexts, a term, and a type, written as $\Delta; \Gamma \vdash M : A$.

[1] This is known to be the *call-by-name* Girard translation (cf. [12]) and we only follow this version in later discussions. However, we conjecture that our work can apply to other versions.

Syntactic category

Types $A, B, C ::= p \mid A \supset B \mid \Box A$

Terms $M, N, L ::= x \mid \lambda x : A.M \mid M\,N \mid \Box M \mid \mathbf{let}\,\Box x = M\,\mathbf{in}\,N$

Reduction rule

$(\beta \supset)\quad (\lambda x : A.M)\,N \rightsquigarrow M[x := N]$

$(\beta\Box)\quad \mathbf{let}\,\Box x = \Box N\,\mathbf{in}\,M \rightsquigarrow M[x := N]$

Typing rule

$$\frac{}{\Delta; \Gamma, x : A \vdash x : A}\,\mathrm{Ax} \qquad \frac{}{\Delta, x : A; \Gamma \vdash x : A}\,\Box\mathrm{Ax}$$

$$\frac{\Delta; \Gamma, x : A \vdash M : B}{\Delta; \Gamma \vdash (\lambda x : A.M) : A \supset B}\supset\mathrm{I} \qquad \frac{\Delta; \Gamma \vdash M : A \supset B \quad \Delta; \Gamma \vdash N : A}{\Delta; \Gamma \vdash M\,N : B}\supset\mathrm{E}$$

$$\frac{\Delta; \emptyset \vdash M : A}{\Delta; \Gamma \vdash \Box M : \Box A}\,\Box\mathrm{I} \qquad \frac{\Delta; \Gamma \vdash M : \Box A \quad \Delta, x : A; \Gamma \vdash N : B}{\Delta; \Gamma \vdash \mathbf{let}\,\Box x = M\,\mathbf{in}\,N : B}\,\Box\mathrm{E}$$

Figure 4 Definition of $\lambda^{\Box}$.

The intuition behind the judgment $\Delta; \Gamma \vdash M : A$ is that the context Δ is intended to implicitly represent assumptions for types of form $\Box A$, while the context Γ is used to represent ordinary assumptions as in the simply-typed λ-calculus.

The typing rules are summarized as follows. Ax, $\supset$I, and $\supset$E are all standard, although they are defined in the dual-context style. $\Box$Ax is another variable rule, which can be seen as what to formalize the modal axiom T (i.e., $\vdash \Box A \supset A$) from the logical viewpoint. $\Box$I is a rule for the constructor of $\Box A$, which corresponds to the necessitation rule for the $\Box$-modality. Similarly, $\Box$E is for the destructor of $\Box A$, which corresponds to the elimination rule.

The *reduction* $\rightsquigarrow$ is defined to be the least compatible relation on terms generated by $(\beta \supset)$ and $(\beta\Box)$. The *multistep reduction* $\rightsquigarrow^{+}$ is defined to be the transitive closure of $\rightsquigarrow$.

3 Linear-logical reconstruction

3.1 Naive attempt at the linear-logical reconstruction

It is natural for a "linear-logical reconstruction" of IS4 to define a system that has both properties of linear logic and modal logic, so as to be a target system for an S4-version of Girard translation. However, a naive combination of linear logic and modal logic is not suitable to establish a faithful translation.

Let us consider what happens if we adopt a naive system. The simplest way to define a target system for the S4-version of Girard translation is to make an extension of IMELL with the $\Box$-modality. Suppose that a deductive system IMELL$^{\Box}$ is such a calculus, that is, the formulae of IMELL$^{\Box}$ are defined by the following grammar:

$$A, B ::= p \mid A \multimap B \mid\, !A \mid \Box A$$

with the inference rules being those of IMELL, along with the rules $\Box$R and $\Box$L of LJ$^{\Box}$.

As in the case of Girard translation from IL to IMELL, we have to establish the following theorem for some translation $\lceil - \rceil$:

If $\Gamma \vdash A$ is derivable in LJ$^{\Box}$, then so is $!\lceil \Gamma \rceil \vdash \lceil A \rceil$ in IMELL$^{\Box}$.

but, if we extend our previous translation $\lceil - \rceil$ from IL to IMELL with $\lceil \Box A \rceil \overset{\mathrm{def}}{=} \Box \lceil A \rceil$, we get stuck in the case of $\Box$R. This is because we need to establish the inference $\Box'$ in Figure 5, which means that we have to be able to obtain a derivation of form $!\lceil \Box\Gamma \rceil \vdash \Box\lceil A \rceil$ from that of $!\lceil \Box\Gamma \rceil \vdash \lceil A \rceil$ in IMELL$^{\Box}$.

$$\begin{array}{c}\vdots\\ \dfrac{\Box\Gamma \vdash A}{\Box\Gamma \vdash \Box A}\,\Box\mathrm{R}\\ \text{in } \mathrm{LJ}^{\Box}\end{array} \quad \overset{\lceil - \rceil}{\longmapsto} \quad \begin{array}{c}\vdots\\ \dfrac{!\lceil\Box\Gamma\rceil \vdash \lceil A\rceil}{!\lceil\Box\Gamma\rceil \vdash \lceil\Box A\rceil}\,\Box'\\ \text{in } \mathrm{IMELL}^{\Box}\end{array}$$

Figure 5 Translation for the case of $\Box$R.

$$\dfrac{\Box(p \supset q), \Box p \vdash q}{\Box(p \supset q), \Box p \vdash \Box q}\,\Box\mathrm{R}$$

Figure 6 Valid inference in $\mathrm{LJ}^{\Box}$.

$$\dfrac{!\Box(!p \multimap q), !\Box p \vdash q}{!\Box(!p \multimap q), !\Box p \vdash \Box q}\,\Box\mathrm{R}$$

Figure 7 Invalid inference in $\mathrm{IMELL}^{\Box}$.

Syntactic category

Formulae $A, B, C ::= p \mid A \multimap B \mid !A \mid \boxed{!}A$

Inference rule

$$\dfrac{}{A \vdash A}\,\mathrm{Ax} \qquad \dfrac{\Gamma \vdash A \quad A, \Gamma' \vdash B}{\Gamma, \Gamma' \vdash B}\,\mathrm{Cut}$$

$$\dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B}\multimap\mathrm{R} \qquad \dfrac{\Gamma \vdash A \quad \Gamma', B \vdash C}{\Gamma, \Gamma', A \multimap B \vdash C}\multimap\mathrm{L} \qquad \dfrac{\boxed{!}\Delta, !\Gamma \vdash A}{\boxed{!}\Delta, !\Gamma \vdash !A}\,!\mathrm{R} \qquad \dfrac{\Gamma, A \vdash B}{\Gamma, !A \vdash B}\,!\mathrm{L} \qquad \dfrac{\boxed{!}\Delta \vdash A}{\boxed{!}\Delta \vdash \boxed{!}A}\,\boxed{!}\mathrm{R}$$

$$\dfrac{\Gamma, A \vdash B}{\Gamma, \boxed{!}A \vdash B}\,\boxed{!}\mathrm{L} \qquad \dfrac{\Gamma \vdash B}{\Gamma, !A \vdash B}\,!\mathrm{W} \qquad \dfrac{\Gamma, !A, !A \vdash B}{\Gamma, !A \vdash B}\,!\mathrm{C} \qquad \dfrac{\Gamma \vdash B}{\Gamma, \boxed{!}A \vdash B}\,\boxed{!}\mathrm{W} \qquad \dfrac{\Gamma, \boxed{!}A, \boxed{!}A \vdash B}{\Gamma, \boxed{!}A \vdash B}\,\boxed{!}\mathrm{C}$$

Figure 8 Definition of $\mathrm{IMELL}^{\boxed{!}}$.

However, the inference $\Box'$ is invalid in $\mathrm{IMELL}^{\Box}$ in general, because there exists a counterexample. First, the inference shown in Figure 6 is valid, and the judgment $\Box(p \supset q), \Box p \vdash \Box q$ is indeed derivable in $\mathrm{LJ}^{\Box}$. However, the corresponding inference via $\lceil - \rceil$ is invalid as Figure 7 shows. In the figure, the judgments correspond to those in Figure 6 via $\lceil - \rceil$, but the inference $\Box$R in Figure 7 is invalid in $\mathrm{IMELL}^{\Box}$ due to the side-condition of $\Box$R. Even worse, we can see that the judgment $!\Box(!p \multimap q), !\Box p \vdash \Box q$ is itself underivable in $\mathrm{IMELL}^{\Box}$[2].

Moreover, one may think the other cases that we extend the original translation $\lceil - \rceil$ from IL to IMELL with $\lceil \Box A \rceil \overset{\mathrm{def}}{=} !\Box\lceil A \rceil$ or $\lceil \Box A \rceil \overset{\mathrm{def}}{=} \Box!\lceil A \rceil$ will work to obtain a faithful translation. However, the judgment $\Box p \vdash \Box\Box p$ will be a counter-example in either case.

All in all, the problem of the naive combination formulated as $\mathrm{IMELL}^{\Box}$ intuitively came from an undesirable interaction between the right rules of the two modalities:

$$\dfrac{!\Gamma \vdash A}{!\Gamma \vdash !A}\,!\mathrm{R} \qquad\qquad \dfrac{\Box\Gamma \vdash A}{\Box\Gamma \vdash \Box A}\,\Box\mathrm{R}$$

Each of these rules has a side-condition: the conclusion $!A$ in !R must be derived from the modalized context $!\Gamma$, and similarly for $\Box A$ in $\Box$R. This makes it hard to obtain a faithful S4-version of Girard translation for this naive extension.

3.2 Modal linear logic

We propose a *modal linear logic* to give a faithful S4-version of Girard translation from IS4.

First of all, the problem we have identified essentially came from the fact that there is no relationship between '!' and '$\Box$', and hence the side-conditions of !R and $\Box$R do not hold when we intuitively expect them to hold. Thus, we introduce a modality, '$\boxed{!}$' combining '!' and '$\Box$', to solve this problem.

[2] Precisely speaking, this can be shown as a consequence of the cut-elimination theorem of $\mathrm{IMELL}^{\Box}$, and the theorem was shown in the authors' previous work [6].

$\lceil A \rceil$	$\lceil \Gamma \rceil$
$\lceil p \rceil \stackrel{\text{def}}{=} p,\ \ \lceil A \supset B \rceil \stackrel{\text{def}}{=} (!\lceil A \rceil) \multimap \lceil B \rceil,\ \ \lceil \Box A \rceil \stackrel{\text{def}}{=} \boxed{!}\lceil A \rceil$	$\lceil \Gamma \rceil \stackrel{\text{def}}{=} \{(x : \lceil A \rceil) \mid (x : A) \in \Gamma\}$

Figure 9 Definition of the S4-version of Girard translation.

Our modal linear logic, which is called $\mathrm{IMELL}^{\boxed{!}}$, is defined by a sequent calculus which is given in Figure 8. As we mentioned, the formulae are defined as an extension of those of IMELL with the $\boxed{!}$-modality. A point is that the !-modality is still there with the $\boxed{!}$-modality.

The $\boxed{!}$-modality is defined so as to have properties of both '!' and '$\Box$', but '!' still behaves similarly to IMELL. Therefore, all the intuitions of the inference rules except !R and $\boxed{!}$R should be clear. The rules !R and $\boxed{!}$R reflect the "strength" between the modalities '!' and '$\boxed{!}$'. Indeed, '!' and '$\boxed{!}$' satisfy the S4 axiomata and '$\boxed{!}$' is stronger than '!'.

▶ **Example 3.** The following hold:

1. $\vdash !A \multimap A$ and $\vdash \boxed{!}A \multimap A$
2. $\vdash !(A \multimap B) \multimap !A \multimap !B$ and $\vdash \boxed{!}(A \multimap B) \multimap \boxed{!}A \multimap \boxed{!}B$
3. $\vdash !A \multimap !!A$ and $\vdash \boxed{!}A \multimap \boxed{!}\boxed{!}A$
4. $\vdash \boxed{!}A \multimap !A$ but $\nvdash !A \multimap \boxed{!}A$

▶ Remark 4. In Example 3, the first three represent the so-called S4 axiomata: *T*, *K*, and *4*. The last one represents the strength of the two modalities. Actually, assuming the !-modality and the $\boxed{!}$-modality to satisfy the S4 axiomata and the "strength" axiom $\vdash \boxed{!}A \multimap !A$ is enough to characterize our modal linear logic (see Section 5 for more details).

The cut-elimination theorem for $\mathrm{IMELL}^{\boxed{!}}$ is shown similarly to the case of IMELL, and hence $\mathrm{IMELL}^{\boxed{!}}$ is consistent. The addition of '$\boxed{!}$' causes no problems in the proof.

▶ **Definition 5** (Cut-degree and degree)**.** *For an application of* Cut *in a proof, its* cut-degree *is defined to be the number of logical connectives in the cut-formula. The* degree *of a proof is defined to be the maximal cut-degree of the proof (and* 0 *if there is no application of* Cut*).*

▶ **Theorem 6** (Cut-elimination)**.** *The rule* Cut *in* $\mathrm{IMELL}^{\boxed{!}}$ *is admissible, i.e., if* $\Gamma \vdash A$ *is derivable, then there is a derivation of the same judgment without any applications of* Cut.

Proof. We follow the proof for propositional linear logic by Lincoln et al. [9]. To show the admissibility of Cut, we consider the admissibility of the following cut rules:

$$\frac{\Gamma \vdash !A \qquad \Gamma', (!A)^n \vdash B}{\Gamma, \Gamma' \vdash B}\ !\mathrm{Cut} \qquad\qquad \frac{\Gamma \vdash \boxed{!}A \qquad \Gamma', (\boxed{!}A)^n \vdash B}{\Gamma, \Gamma' \vdash B}\ \boxed{!}\mathrm{Cut}$$

where $(C)^n$ denotes the multiset that has n occurrences of C and n is assumed to be positive as a side-condition; and Γ' in !Cut (resp. in $\boxed{!}$Cut) is supposed to contain no formulae of form $!A$ (resp. $\boxed{!}A$). The *cut-degrees* of !Cut and $\boxed{!}$Cut are defined similarly to that of Cut.

Then, all the three rules (Cut, !Cut, $\boxed{!}$Cut) are shown to be admissible by simultaneous induction on the lexicographic complexity $\langle \delta, h \rangle$, where δ is the degree of the assumed derivation and h is its height. See the appendix for details of the proof. ◀

▶ **Corollary 7** (Consistency)**.** $\mathrm{IMELL}^{\boxed{!}}$ *is consistent, i.e., there exists an underivable judgment.*

Then, we can define an S4-version of Girard translation as in Figure 9, and it can be justified by the following theorem, which is readily shown by induction on the derivation.

▶ **Theorem 8** (Soundness)**.** *If* $\Box\Delta, \Gamma \vdash A$ *in* $\mathrm{LJ}^{\Box}$*, then* $\boxed{!}\lceil \Delta \rceil, !\lceil \Gamma \rceil \vdash \lceil A \rceil$ *in* $\mathrm{IMELL}^{\boxed{!}}$.

Syntactic category

Types $A, B, C ::= p \mid A \multimap B \mid\, !A \mid \boxed{!}A$

Terms $M, N, L ::= x \mid \lambda x : A.M \mid M\,N \mid\, !M \mid \boxed{!}M$
$\mid \textbf{let}\, !x = M \,\textbf{in}\, N \mid \textbf{let}\ \boxed{!}x = M \,\textbf{in}\, N$

Reduction rule

$(\beta \multimap)\quad (\lambda x : A.M)\,N \rightsquigarrow M[x := N]$

$(\beta !)\quad \textbf{let}\, !x = !N \,\textbf{in}\, M \rightsquigarrow M[x := N]$

$(\beta \boxed{!})\quad \textbf{let}\ \boxed{!}x = \boxed{!}N \,\textbf{in}\, M \rightsquigarrow M[x := N]$

Typing rule

$$\frac{}{\Delta; \Gamma; x : A \vdash x : A}\,\text{LinAx} \qquad \frac{}{\Delta; \Gamma, x : A; \emptyset \vdash x : A}\,!\text{Ax} \qquad \frac{}{\Delta, x : A; \Gamma; \emptyset \vdash x : A}\,\boxed{!}\text{Ax}$$

$$\frac{\Delta; \Gamma; \Sigma, x : A \vdash M : B}{\Delta; \Gamma; \Sigma \vdash \lambda x : A.M : A \multimap B}\,\multimap\text{I} \qquad \frac{\Delta; \Gamma; \Sigma \vdash M : A \multimap B \qquad \Delta; \Gamma; \Sigma' \vdash N : A}{\Delta; \Gamma; \Sigma, \Sigma' \vdash M\,N : B}\,\multimap\text{E}$$

$$\frac{\Delta; \Gamma; \emptyset \vdash M : A}{\Delta; \Gamma; \emptyset \vdash !M : !A}\,!\text{I} \qquad \frac{\Delta; \Gamma; \Sigma \vdash M : !A \qquad \Delta; \Gamma, x : A; \Sigma' \vdash N : B}{\Delta; \Gamma; \Sigma, \Sigma' \vdash \textbf{let}\, !x = M \,\textbf{in}\, N : B}\,!\text{E}$$

$$\frac{\Delta; \emptyset; \emptyset \vdash M : A}{\Delta; \Gamma; \emptyset \vdash \boxed{!}M : \boxed{!}A}\,\boxed{!}\text{I} \qquad \frac{\Delta; \Gamma; \Sigma \vdash M : \boxed{!}A \qquad \Delta, x : A; \Gamma; \Sigma' \vdash N : B}{\Delta; \Gamma; \Sigma, \Sigma' \vdash \textbf{let}\ \boxed{!}x = M \,\textbf{in}\, N : B}\,\boxed{!}\text{E}$$

Figure 10 Definition of $\lambda^{\boxed{!}}$.

4 Curry–Howard correspondence

In this section, we give a computational interpretation for our modal linear logic through the Curry–Howard correspondence and establish the corresponding S4-version of Girard translation for the modal linear logic in terms of typed λ-calculus.

4.1 Typed λ-calculus for the intuitionistic modal linear logic

We introduce $\lambda^{\boxed{!}}$ (pronounced by "lambda bangbox") that is a typed λ-calculus corresponding to the modal linear logic under the Curry–Howard correspondence. The calculus $\lambda^{\boxed{!}}$ can be seen as an integration of $\lambda^{\Box}$ of Pfenning and Davies and the linear λ-calculus for dual intuitionistic linear logic of Barber [2]. The rules of $\lambda^{\boxed{!}}$ are designed considering the "necessity" of modal logic and the "linearity" of linear logic, and formally defined as in Figure 10.

The structure of types are exactly the same as that of formulae in IMELL$^{\boxed{!}}$. Terms are defined as an extension of the simply-typed λ-calculus with the following: the terms $!M$ and $\textbf{let}\, !x = M \,\textbf{in}\, N$, which are a constructor and a destructor for types $!A$, respectively; and the terms $\boxed{!}M$ and $\textbf{let}\ \boxed{!}x = M \,\textbf{in}\, N$, which are those for types $\boxed{!}A$ similarly. Note that the variable x in $\textbf{let}\, !x = M \,\textbf{in}\, N$ and $\textbf{let}\ \boxed{!}x = M \,\textbf{in}\, N$ is supposed to be *bound*.

A *(type) context* is defined by the same way as $\lambda^{\Box}$ and a *(type) judgment* consists of three contexts, a term and a type, written as $\Delta; \Gamma; \Sigma \vdash M : A$. These three contexts of a judgment $\Delta; \Gamma; \Sigma \vdash M : A$ have the following intuitive meaning: (1) Δ implicitly represents a context for modalized types of form $\boxed{!}A$; (2) Γ implicitly represents a context for modalized types of form $\Box A$; (3) Σ represents an ordinary context but its elements must be used linearly.

The intuitive meanings of the typing rules are as follows. Each of the first three rules is a variable rule depending on the context's kind. It is allowed for the Δ-part and the Γ-part to weaken the antecedent in these rules, but is not for the Σ-part since it must satisfy the linearity condition. The rules $\multimap$I and $\multimap$E are for the type $\multimap$, and again, the $\multimap$E is designed to satisfy the linearity. The remaining rules are for types $!A$ and $\boxed{!}A$.

The *reduction* $\rightsquigarrow$ is defined to be the least compatible relation on terms generated by $(\beta \multimap)$, $(\beta !)$, and $(\beta \boxed{!})$. The *multistep reduction* $\rightsquigarrow^{+}$ is defined as in the case of $\lambda^{\Box}$.

Then, we can show the subject reduction and the strong normalization of $\lambda^{\boxed{!}}$ as follows.

▶ **Lemma 9** (Substitution).

1. *If* $\Delta; \Gamma; \Sigma, x : A \vdash M : B$ *and* $\Delta; \Gamma; \Sigma' \vdash N : A$*, then* $\Delta; \Gamma; \Sigma, \Sigma' \vdash M[x := N] : B$*;*
2. *If* $\Delta; \Gamma, x : A; \Sigma \vdash M : B$ *and* $\Delta; \Gamma; \emptyset \vdash N : A$*, then* $\Delta; \Gamma; \Sigma \vdash M[x := N] : B$*;*
3. *If* $\Delta, x : A; \Gamma; \Sigma \vdash M : B$ *and* $\Delta; \emptyset; \emptyset \vdash N : A$*, then* $\Delta; \Gamma; \Sigma \vdash M[x := N] : B$*.*

▶ **Theorem 10** (Subject reduction). *If* $\Delta; \Gamma; \Sigma \vdash M : A$ *and* $M \rightsquigarrow N$*, then* $\Delta; \Gamma; \Sigma \vdash N : A$*.*

Proof. By induction on the derivation of $\Delta; \Gamma; \Sigma \vdash M : A$ together with Lemma 9. ◀

▶ **Theorem 11** (Strong normalization). *For well-typed term* M*, there are no infinite reduction sequences starting from* M*.*

Proof. By embedding to a typed λ-calculus of the $(!, \multimap)$-fragment of dual intuitionistic linear logic, named $\lambda^{!,\multimap}$, which is shown to be strongly normalizing by Ohta and Hasegawa [16].

The details are in the appendix, but the intuition is described as follows. First, for every well-typed term M, we define the term $(M)^{\ddagger}$ by replacing the occurrences of $\boxed{!}N$ and $\mathbf{let}\ \boxed{!}x = N\,\mathbf{in}\,L$ in M with $!(N)^{\ddagger}$ and $\mathbf{let}\,!x = (N)^{\ddagger}\,\mathbf{in}\,(L)^{\ddagger}$, respectively. Then, we can show that $(M)^{\ddagger}$ is typable in $\lambda^{!,\multimap}$, because the structure of '$\boxed{!}$' collapses to that of '!', and that the embedding $(-)^{\ddagger}$ preserves reductions. Therefore, $\lambda^{\boxed{!}}$ is strongly normalizing. ◀

As we mentioned, we can view that $\lambda^{\boxed{!}}$ is indeed a typed λ-calculus for the intuitionistic modal linear logic. A natural deduction that corresponds to $\lambda^{\boxed{!}}$ is obtained as the "logical-part" of the calculus, and we can show that the natural deduction is equivalent to $\mathrm{IMELL}^{\boxed{!}}$.

▶ **Definition 12** (Natural deduction). *A natural deduction for modal linear logic, called* $\mathrm{NJ}^{\boxed{!}}$*, is defined to be one that is extracted from* $\lambda^{\boxed{!}}$ *by erasing term annotations.*

▶ **Fact 13** (Curry–Howard correspondence). *There is a one-to-one correspondence between* $\mathrm{NJ}^{\boxed{!}}$ *and* $\lambda^{\boxed{!}}$*, which preserves provability/typability and proof-normalizability/reducibility.*

▶ **Lemma 14** (Judgmental reflection). *The following hold in* $\mathrm{NJ}^{\boxed{!}}$*.*

1. $\Delta; \Gamma; \Sigma, !A \vdash B$ *if and only if* $\Delta; \Gamma, A; \Sigma \vdash B$*;*
2. $\Delta; \Gamma; \Sigma, \boxed{!}A \vdash B$ *if and only if* $\Delta, A; \Gamma; \Sigma \vdash B$*.*

▶ **Theorem 15** (Equivalence). $\Delta; \Gamma; \Sigma \vdash A$ *in* $\mathrm{NJ}^{\boxed{!}}$ *if and only if* $\boxed{!}\Delta, !\Gamma, \Sigma \vdash A$ *in* $\mathrm{IMELL}^{\boxed{!}}$*.*

Proof. By straightforward induction. Lemma 14 is used to show the if-part. ◀

4.2 Embedding from the modal λ-calculus by Pfenning and Davies

We give a translation from Pfenning and Davies' $\lambda^{\Box}$ to our $\lambda^{\boxed{!}}$. We also show that the translation preserves the reductions of $\lambda^{\Box}$, and thus it can be seen as the S4-version of Girard translation on the level of proofs through the Curry–Howard correspondence.

To give the translation, we introduce two meta λ-terms in $\lambda^{\boxed{!}}$ to encode the function space $\supset$ of $\lambda^{\Box}$. The simulation of reduction of $(\lambda x : A.M)\,N$ in $\lambda^{\Box}$ can be shown readily.

▶ **Definition 16.** *Let* M *and* N *be terms such that* $\Delta; \Gamma, x : A; \Sigma \vdash M : B$ *and* $\Delta; \Gamma; \emptyset \vdash N : A$*. Then,* $\overline{\lambda} x : A.M$ *and* $M \overline{@} N$ *are defined as the terms* $\lambda y :!A.\mathbf{let}\,!x = y\,\mathbf{in}\,M$ *and* $M\,(!N)$*, respectively, where* y *is chosen to be fresh, i.e., it is a variable satisfying* $y \notin (\mathrm{FV}(M) \cup \{x\})$*.*

▶ **Lemma 17** (Derivable full-function space). *The following rules are derivable in* $\lambda^{\boxed{!}}$*:*

$$\frac{\Delta; \Gamma, x : A; \Sigma \vdash M : B}{\Delta; \Gamma; \Sigma \vdash (\overline{\lambda} x : A.M) :!A \multimap B} \qquad \frac{\Delta; \Gamma; \Sigma \vdash M :!A \multimap B \qquad \Delta; \Gamma; \emptyset \vdash N : A}{\Delta; \Gamma; \Sigma \vdash M \overline{@} N : B}$$

Moreover, it holds that $(\overline{\lambda} x : A.M)\overline{@}N \rightsquigarrow^{+} M[x := N]$ *in* $\lambda^{\boxed{!}}$*.*

$\lceil A \rceil$	$\mathcal{T}\lceil M \rceil$
$\lceil p \rceil \stackrel{\text{def}}{=} p$	$\mathcal{T}\lceil x \rceil \stackrel{\text{def}}{=} x$
$\lceil A \supset B \rceil \stackrel{\text{def}}{=} !\lceil A \rceil \multimap \lceil B \rceil$	$\mathcal{T}\lceil \lambda x : A.M \rceil \stackrel{\text{def}}{=} \overline{\lambda} x : \lceil A \rceil.\mathcal{T}\lceil M \rceil$
$\lceil \Box A \rceil \stackrel{\text{def}}{=} \boxed{!}\lceil A \rceil$	$\mathcal{T}\lceil M\ N \rceil \stackrel{\text{def}}{=} \mathcal{T}\lceil M \rceil \overline{@} \mathcal{T}\lceil N \rceil$
$\lceil \Gamma \rceil$	$\mathcal{T}\lceil \Box M \rceil \stackrel{\text{def}}{=} \boxed{!}\mathcal{T}\lceil M \rceil$
$\lceil \Gamma \rceil \stackrel{\text{def}}{=} \{(x : \lceil A \rceil) \mid (x : A) \in \Gamma\}$	$\mathcal{T}\lceil \mathbf{let}\ \Box x = M\ \mathbf{in}\ N \rceil \stackrel{\text{def}}{=} \mathbf{let}\ \boxed{!} x = \mathcal{T}\lceil M \rceil\ \mathbf{in}\ \mathcal{T}\lceil N \rceil$

Figure 11 Definitions of the S4-version of Girard translation in term of typed λ-calculus.

Together with the above meta λ-terms $\overline{\lambda} x : A.M$ and $M \overline{@} N$, we can define the translation from $\lambda^{\Box}$ into $\lambda^{\boxed{!}}$ and show that it preserves typability and reducibility.

▶ **Definition 18** (Translation). *The* translation *from $\lambda^{\Box}$ to $\lambda^{\boxed{!}}$ is defined to be the triple of the type/context/term translations $\lceil A \rceil$, $\lceil \Gamma \rceil$, and $\mathcal{T}\lceil M \rceil$ defined in Figure 11.*

▶ **Theorem 19** (Embedding). *$\lambda^{\Box}$ can be embedded into $\lambda^{\boxed{!}}$, i.e., the following hold:*
1. *If $\Delta; \Gamma \vdash M : A$ in $\lambda^{\Box}$, then $\lceil \Delta \rceil; \lceil \Gamma \rceil; \emptyset \vdash \mathcal{T}\lceil M \rceil : \lceil A \rceil$ in $\lambda^{\boxed{!}}$.*
2. *If $M \rightsquigarrow M'$ in $\lambda^{\Box}$, then $\mathcal{T}\lceil M \rceil \rightsquigarrow^{+} \mathcal{T}\lceil M' \rceil$ in $\lambda^{\boxed{!}}$.*

Proof. By induction on the derivation of $\Delta; \Gamma \vdash M : A$ and $M \rightsquigarrow M'$ in $\lambda^{\Box}$, respectively. ◀

From the logical point of view, Theorem 19.1 can be seen as another S4-version of Girard translation (in the style of natural deduction) that corresponds to Theorem 8; and Theorem 19.2 gives a justification that the S4-version of Girard translation is correct with respect to the level of proofs, i.e., it preserves proof-normalizations as well as provability.

5 Axiomatization of modal linear logic

We give an axiomatic characterization of the intuitionistic modal linear logic. To do so, we define a typed combinatory logic, called $\mathrm{CL}^{\boxed{!}}$, which can be seen as a Hilbert-style deductive system of modal linear logic through the Curry–Howard correspondence. In this section, we only aim to provide the equivalence between $\mathrm{NJ}^{\boxed{!}}$ and the Hilbert-style, while $\mathrm{CL}^{\boxed{!}}$ satisfies several desirable properties, e.g., the subject reduction and the strong normalizability.

The definition of $\mathrm{CL}^{\boxed{!}}$ is given in Figure 12. The set of types has the same structure as that in $\lambda^{\boxed{!}}$. A term is either a *variable*, a *combinator*, a necessitated term by '!', or a necessitated term by '$\boxed{!}$'. The notions of *(type) context* and *(type) judgment* are defined similarly to those of $\lambda^{\boxed{!}}$.

Every combinator c has its type as defined in the list in the figure, and is denoted by typeof(c). Then, the typing rules are described as follows: Ax and MP are the standard rules, which logically correspond to an axiom rule of the set of axiomata, and *modus ponens*, respectively. The others are defined by the same way as in $\lambda^{\boxed{!}}$.

The *reduction* $\rightsquigarrow$ of combinators is defined to be the least compatible relation on terms generated by the reduction rules listed in the figure.

▶ Remark 20. $\mathrm{CL}^{\boxed{!}}$ can be seen as an extension of *linear combinatory algebra* of Abramsky et al. [1] with the $\boxed{!}$-modality, or equivalently, a linear-logical reconstruction of Pfenning's modally-typed combinatory logic [17]. The combinators $\mathrm{T}^{!}$, $\mathrm{D}^{!}$, $4^{!}$ represent the S4 axiomata for the !-modality, and similarly, $\mathrm{T}^{\boxed{!}}$, $\mathrm{D}^{\boxed{!}}$, $4^{\boxed{!}}$ represent those for the $\boxed{!}$-modality. E is the only one combinator to characterize the strength between the two modalities.

As we defined $\mathrm{NJ}^{\boxed{!}}$ from $\lambda^{\boxed{!}}$, we can define the Hilbert-style deductive system (with open assumptions) for the intuitionistic modal linear logic via $\mathrm{CL}^{\boxed{!}}$.

Syntactic category

Types $A, B, C ::= p \mid A \multimap B \mid !A \mid \boxed{!}A$

Terms $M, N, L ::= x \mid c \mid !M \mid \boxed{!}M$

Typing rule

$$\frac{(c \text{ is a combinator})}{\Delta;\Gamma;\emptyset \vdash c : \mathrm{typeof}(c)}\,\mathrm{Ax} \qquad \frac{\Delta;\Gamma;\Sigma \vdash M : A \multimap B \qquad \Delta;\Gamma;\Sigma' \vdash N : A}{\Delta;\Gamma;\Sigma,\Sigma' \vdash M\,N : B}\,\mathrm{MP}$$

$$\frac{}{\Delta;\Gamma;x : A \vdash x : A}\,\mathrm{LinAx} \qquad \frac{}{\Delta;\Gamma,x : A;\emptyset \vdash x : A}\,!\mathrm{Ax} \qquad \frac{}{\Delta,x : A;\Gamma;\emptyset \vdash x : A}\,\boxed{!}\mathrm{Ax}$$

$$\frac{\Delta;\Gamma;\emptyset \vdash M : A}{\Delta;\Gamma;\emptyset \vdash !M : !A}\,! \qquad \frac{\Delta;\emptyset;\emptyset \vdash M : A}{\Delta;\Gamma;\emptyset \vdash \boxed{!}M : \boxed{!}A}\,\boxed{!}$$

Combinator

- $\vdash \mathrm{I} : A \multimap A$
- $\vdash \mathrm{B} : (B \multimap C) \multimap (A \multimap B) \multimap A \multimap C$
- $\vdash \mathrm{C} : (A \multimap B \multimap C) \multimap B \multimap A \multimap C$
- $\vdash \mathrm{S}^\delta : (\delta A \multimap B \multimap C) \multimap (\delta A \multimap B) \multimap \delta A \multimap C$
- $\vdash \mathrm{K}^\delta : A \multimap \delta B \multimap A$
- $\vdash \mathrm{W}^\delta : (\delta A \multimap \delta A \multimap B) \multimap \delta A \multimap B$
- $\vdash \mathrm{T}^\delta : \delta A \multimap A$
- $\vdash \mathrm{D}^\delta : \delta(A \multimap B) \multimap \delta A \multimap \delta B$
- $\vdash 4^\delta : \delta A \multimap \delta\delta A$
- $\vdash \mathrm{E} : \boxed{!}A \multimap !A$

where $\delta \in \{!, \boxed{!}\}$

Reduction

$\mathrm{I}\,M$	$\rightsquigarrow$	M
$\mathrm{B}\,M\,N\,L$	$\rightsquigarrow$	$M\,(N\,L)$
$\mathrm{C}\,M\,N\,L$	$\rightsquigarrow$	$M\,L\,N$
$\mathrm{S}^\delta\,M\,N\,(\delta L)$	$\rightsquigarrow$	$M\,(\delta L)\,(N\,(\delta L))$
$\mathrm{K}^\delta\,M\,(\delta N)$	$\rightsquigarrow$	M
$\mathrm{W}^\delta\,M\,(\delta N)$	$\rightsquigarrow$	$M\,(\delta N)\,(\delta N)$
$\mathrm{T}^\delta\,(\delta M)$	$\rightsquigarrow$	M
$\mathrm{D}^\delta\,(\delta M)\,(\delta N)$	$\rightsquigarrow$	$\delta(M\,N)$
$4^\delta\,(\delta M)$	$\rightsquigarrow$	$\delta\delta M$
$\mathrm{E}\,\boxed{!}M$	$\rightsquigarrow$	$!M$

where $\delta \in \{!, \boxed{!}\}$

Figure 12 Definition of $\mathrm{CL}^{\boxed{!}}$.

▶ **Definition 21** (Hilbert-style). *A Hilbert-style deductive system for modal linear logic, called* $\mathrm{HJ}^{\boxed{!}}$*, is defined to be one that is extracted from* $\mathrm{CL}^{\boxed{!}}$ *by erasing term annotations.*

▶ **Fact 22** (Curry–Howard correspondnece). *There is a one-to-one correspondence between* $\mathrm{HJ}^{\boxed{!}}$ *and* $\mathrm{CL}^{\boxed{!}}$*, which preserves provability/typability and proof-normalizability/reducibility.*

The deduction theorem of $\mathrm{HJ}^{\boxed{!}}$ can be obtained as a consequence of the so-called *bracket abstraction* of $\mathrm{CL}^{\boxed{!}}$ through Fact 22, which allows us to show the equivalence between $\mathrm{HJ}^{\boxed{!}}$ and $\mathrm{NJ}^{\boxed{!}}$. Therefore, the modal linear logic is indeed axiomatized by $\mathrm{HJ}^{\boxed{!}}$.

▶ **Theorem 23** (Deduction theorem).

1. *If* $\Delta;\Gamma;\Sigma, x : A \vdash M : B$*, then* $\Delta;\Gamma;\Sigma \vdash (\lambda_* x.M) : (A \multimap B)$*;*
2. *If* $\Delta;\Gamma, x : A;\Sigma \vdash M : B$*, then* $\Delta;\Gamma;\Sigma \vdash (\lambda^!_* x.M) : (!A \multimap B)$*;*
3. *If* $\Delta, x : A;\Gamma;\Sigma \vdash M : B$*, then* $\Delta;\Gamma;\Sigma \vdash (\lambda^{\boxed{!}}_* x.M) : (\boxed{!}A \multimap B)$*.*

where $(\lambda_* x.M)$*,* $(\lambda^!_* x.M)$*,* $(\lambda^{\boxed{!}}_* x.M)$ *are bracket abstraction operations that take a variable* x *and a* $\mathrm{CL}^{\boxed{!}}$*-term* M *and returns a* $\mathrm{CL}^{\boxed{!}}$*-term, and the definitions are given in the appendix.*

Proof. By induction on the derivation. The proof is just a type-checking of the result of the bracket abstraction operations. ◀

▶ **Theorem 24** (Equivalence). $\Delta;\Gamma;\Sigma \vdash A$ *in* $\mathrm{HJ}^{\boxed{!}}$ *if and only if* $\Delta;\Gamma;\Sigma \vdash A$ *in* $\mathrm{NJ}^{\boxed{!}}$*.*

Proof. By straightforward induction. We use Theorem 23 and Fact 22 to show the if-part. ◀

▶ **Corollary 25.** $\mathrm{IMELL}^{\boxed{!}}$*,* $\mathrm{NJ}^{\boxed{!}}$*, and* $\mathrm{HJ}^{\boxed{!}}$ *are equivalent with respect to provability.*

Syntactic category	**Inference rule**
Formulae $A, B, C ::= p \mid p^{\perp} \mid A \otimes B \mid A \parr B \mid\, !A \mid\, ?A \mid \boxed{!}A \mid \Diamond\!\!\!?A$	$\dfrac{}{\vdash A^{\perp}, A}\,\mathrm{Ax}$

$$\dfrac{\vdash \Gamma, A \quad \vdash A^{\perp}, \Gamma'}{\vdash \Gamma, \Gamma'}\,\mathrm{Cut} \quad \dfrac{\vdash \Gamma, A \quad \vdash \Gamma', B}{\vdash \Gamma, \Gamma', A \otimes B}\,\otimes \quad \dfrac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B}\,\parr \quad \dfrac{\vdash \Diamond\!\!\!?\Delta, ?\Gamma, A}{\vdash \Diamond\!\!\!?\Delta, ?\Gamma, !A}\,! \quad \dfrac{\vdash \Gamma, A}{\vdash \Gamma, ?A}\,?$$

$$\dfrac{\vdash \Diamond\!\!\!?\Delta, A}{\vdash \Diamond\!\!\!?\Delta, \boxed{!}A}\,\boxed{!} \quad \dfrac{\vdash \Gamma, A}{\vdash \Gamma, \Diamond\!\!\!?A}\,\Diamond\!\!\!? \quad \dfrac{\vdash \Gamma}{\vdash \Gamma, ?A}\,?\mathrm{W} \quad \dfrac{\vdash \Gamma, ?A, ?A}{\vdash \Gamma, ?A}\,?\mathrm{C} \quad \dfrac{\vdash \Gamma}{\vdash \Gamma, \Diamond\!\!\!?A}\,\Diamond\!\!\!?\mathrm{W} \quad \dfrac{\vdash \Gamma, \Diamond\!\!\!?A, \Diamond\!\!\!?A}{\vdash \Gamma, \Diamond\!\!\!?A}\,\Diamond\!\!\!?\mathrm{C}$$

Figure 13 Definition of $\mathrm{CMELL}^{\boxed{!}}$.

6 Geometry of Interaction Machine

In this section, we show a dynamic semantics, called *context semantics*, for the modal linear logic in the style of geometry of interaction machine [10, 11]. As in the usual linear logic, we first define a notion of *proof net* and then define the machine as a token-passing system over those proof nets. Thanks to the simplicity of our logic, the definitions are mostly straightforward extension of those for classical MELL (CMELL).

6.1 Sequent calculus for classical modal linear logic

We define a sequent calculus of classical modal linear logic, called $\mathrm{CMELL}^{\boxed{!}}$. The reason why we define it in the classical setting is for ease of defining the proof nets in the latter part.

Figure 13 shows the definition of $\mathrm{CMELL}^{\boxed{!}}$. The set of formulae are defined as an extension of CMELL-formulae with the two modalities '$\boxed{!}$' and '$\Diamond\!\!\!?$'. A *dual formula* of A, written $A^{\perp}$, is defined by the standard dual formulae in CMELL along with $(\boxed{!}A)^{\perp} \stackrel{\mathrm{def}}{=} \Diamond\!\!\!?(A^{\perp})$ and $(\Diamond\!\!\!?A)^{\perp} \stackrel{\mathrm{def}}{=} \boxed{!}(A^{\perp})$. Here, the $\Diamond\!\!\!?$-modality is the dual of the $\boxed{!}$-modality by definition, and it can be seen as an integration of the ?-modality and the $\Diamond$-modality. The *linear implication* $A \multimap B$ is defined as $A^{\perp} \parr B$ as usual. The inference rules are defined as a simple extension of $\mathrm{IMELL}^{\boxed{!}}$ to the classical setting in the style of "one-sided" sequent.

Then, the cut-elimination theorem for $\mathrm{CMELL}^{\boxed{!}}$ can be shown similarly to the case of $\mathrm{IMELL}^{\boxed{!}}$, and we can see that there exists a trivial embedding from $\mathrm{IMELL}^{\boxed{!}}$ to $\mathrm{CMELL}^{\boxed{!}}$.

▶ **Theorem 26** (Cut-elimination). *The rule* Cut *in* $\mathrm{CMELL}^{\boxed{!}}$ *is admissible.*

▶ **Theorem 27** (Embedding). *If* $\Gamma \vdash A$ *in* $\mathrm{IMELL}^{\boxed{!}}$, *then* $\vdash \Gamma^{\perp}, A$ *in* $\mathrm{CMELL}^{\boxed{!}}$.

6.2 Proof-nets formalization

First, we define *proof structures* for $\mathrm{CMELL}^{\boxed{!}}$. The *proof nets* are then defined to be those proof structures satisfying a condition called *correctness criterion*. Intuitively, a proof net corresponds to an (equivalence class of) proof in $\mathrm{CMELL}^{\boxed{!}}$.

▶ **Definition 28.** *A* node *is one of the graph-theoretic node shown in Figure 14 equipped with* $\mathrm{CMELL}^{\boxed{!}}$ *types on the edges. They are all directed from top to bottom: for example, the* $\parr$ *node has two incoming edges and one outgoing edge. A* !*-node (resp.* $\boxed{!}$*-node) has one outgoing edge typed by* $!A$ *(resp.* $\boxed{!}A$*) and arbitrarily many (possibly zero) outgoing edges typed by* $?A_i$ *and* $\Diamond\!\!\!?B_i$ *(resp.* $\Diamond\!\!\!?A_i$*).*

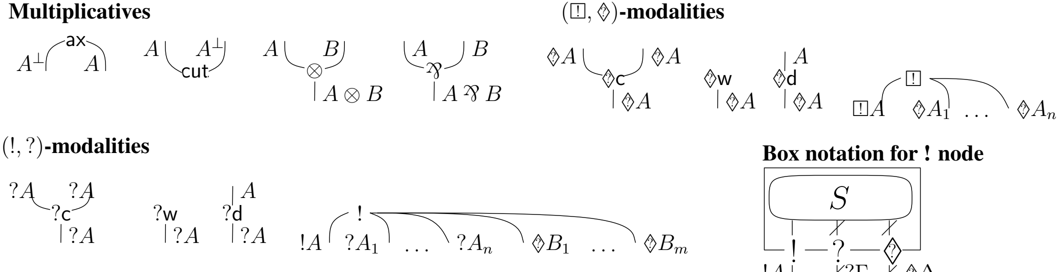


Figure 14 Nodes of proof net and box notation.

A proof structure *is a finite directed graph that satisfies the following conditions:*

- *each edge is with a type that matches the types specified by the nodes (in Figure 14) it is connected to;*
- *some edges may not be connected to any node (called* dangling edges*). Those dangling edges and also the types on those edges are called the* conclusions of the structure*;*
- *the graph is associated with a total map from all the* $!$*-nodes and* $\boxed{!}$*-nodes in it to proof structures called the* contents *of the* $!/\boxed{!}$*-nodes. The map satisfies that the types of the conclusions of a* $!$*-node (resp.* $\boxed{!}$*-node) coincide with the conclusions of its content.*

▶ Remark 29. Formally, a !-node (resp. a $\boxed{!}$-node) and its content are distinctive objects and they are not connected as a directed graph. Though, it is convenient to depict them as if the !-node (resp. $\boxed{!}$-node) represents a "box" filled with its content, as shown at the bottom-right of Figure 14. We also depict multiple edges by an edge with a diagonal line. In what follows, we adopt this "box" notation and multiple edges notation without explicit note.

▶ **Definition 30.** *Given a proof structure S, a* switching path *is an undirected path on S (meaning that the path is allowed to traverse an edge forward or backward) satisfying that on each $\parr$ node, ?c node, and $\Diamond$c node, the path uses at most one of the premises, and that the path uses any edge at most once.*

▶ **Definition 31.** *The* correctness criterion *is the following condition: given a proof structure S, switching paths of S and all contents of* $!$*-nodes,* $\boxed{!}$*-nodes in S are all acyclic and connected. A proof structure satisfying the correctness criterion is called a* proof net*.*

As a counterpart of cut-elimination process in CMELL$^{\boxed{!}}$, the notion of *reduction* is defined for proof structures (and hence for proof nets): this intuition is made precise by Lemma 34 where $(-)^\bullet$ is the translation from CMELL$^{\boxed{!}}$ to proof nets, whose definition is omitted here since it is defined analogously to that of CMELL and CMELL proof net. The lemmata below are naturally obtained by extending the case for CMELL since the $\boxed{!}$-modality has mostly the same logical structure as the !-modality.

▶ **Definition 32.** Reductions *of proof structures are local graph reductions defined by the set of rules depicted in Figure 15.*

▶ **Lemma 33.** *Let $S \to S'$ be a reduction between proof structures. If S is a proof net (i.e., satisfies the correctness criterion), so is S'.*

▶ **Lemma 34.** *Let Π be a proof of $\vdash \Diamond\Delta^\perp, ?\Gamma^\perp, \Sigma^\perp, A$ and suppose that Π reduces to another proof Π'. Then there is a sequence of reductions $(\Pi)^\bullet \to^* (\Pi')^\bullet$ between the proof nets.*

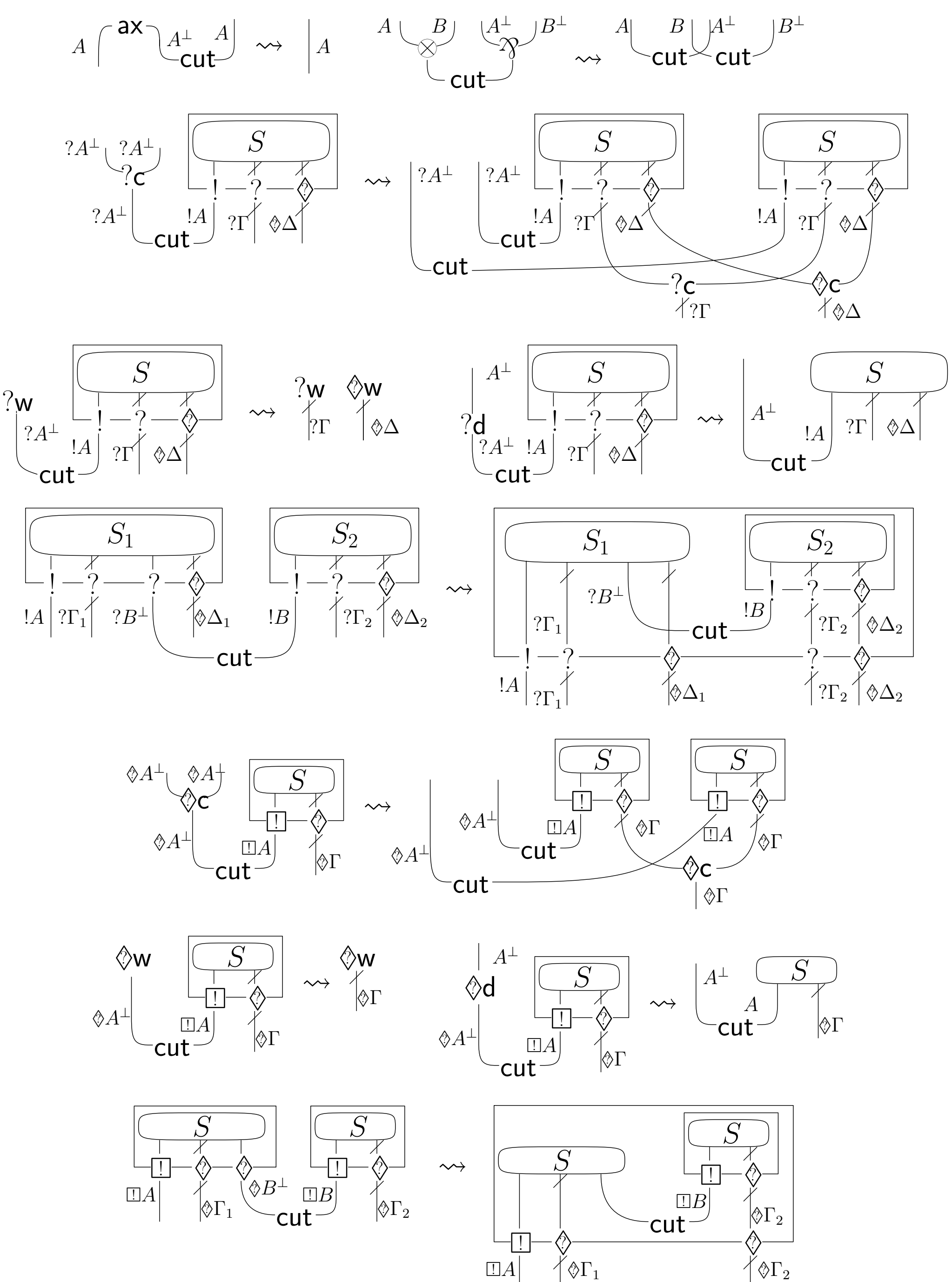


Figure 15 Reduction rules.

6.3 Computational interpretation

▶ **Definition 35.** *A* context *is a triple* $(\mathcal{M}, \mathcal{B}, \mathcal{N})$ *where* $\mathcal{M}, \mathcal{B}, \mathcal{N}$ *are generated by the following grammar:*

$$\mathcal{M} ::= \varepsilon \mid l.\mathcal{M} \mid r.\mathcal{M} \quad \mathcal{B} ::= \varepsilon \mid L.\mathcal{B} \mid R.\mathcal{B} \mid \langle \mathcal{B}, \mathcal{B} \rangle \mid \star \quad \mathcal{N} ::= \varepsilon \mid L'.\mathcal{N} \mid R'.\mathcal{N} \mid \langle \mathcal{N}, \mathcal{N} \rangle \mid \star$$

The intuition of a context is an intermediate state while "evaluating" the proof net (and, by translating into a proof net, a term in $\lambda^{\boxed{!}}$). The geometry of interaction machine calculates the semantic value of a net by traversing the net from a conclusion to another; to traverse the net in a "right way" (more precisely, in a way invariant under net reduction), the context accumulates the information about the path that is already passed. Then, *how* the net is traversed is defined by the notion of *path* over a proof net as we define below.

▶ **Definition 36.** *The* extended dynamic algebra $\Lambda^{\Box *}$ *is a single-sorted* Σ *algebra that contains* $0, 1, p, q, r, r', s, s', t, t', d, d' \colon \Sigma$ *as constants, has an associative operator* $\cdot \colon \Sigma \times \Sigma \to \Sigma$ *and operators* $(-)^* \colon \Sigma \to \Sigma$*,* $! \colon \Sigma \to \Sigma$*,* $\boxed{!} \colon \Sigma \to \Sigma$*, equipped with a formal sum* $+$*, and satisfies the equations below. Hereafter, we write* xy *for* $x \cdot y$ *where* x *and* y *are metavariables over* Σ*.*

$$\begin{array}{rlrl}
0^* = !0 = 0 & 1^* = !1 = 1 & 0x = x0 = 0 & 1x = x1 = x \\
!(x)^* = !(x^*) & (xy)^* = y^* x^* & (x^*)^* = x & !(x)!(y) = !(xy) \\
\boxed{!}(x)\,\boxed{!}(y) = \boxed{!}(xy) & p^* p = q^* q = 1 & q^* p = p^* q = 0 & r^* r = s^* s = 1 \\
s^* r = r^* s = 0 & d^* d = 1 & t^* t = 1 & p'^* p' = q'^* q' = 1 \\
q'^* p' = p'^* q' = 0 & r'^* r' = s'^* s' = 1 & s'^* r' = r'^* s' = 0 & d'^* d' = 1 \\
t'^* t' = 1 & !(x) r = r !(x) & !(x) s = s !(x) & !(x) t = t !!(x) \\
!(x) d = dx & \boxed{!}(x) r' = r'\,\boxed{!}(x) & \boxed{!}(x) s' = s'\,\boxed{!}(x) & \boxed{!}(x) t' = t'\,\boxed{!}\boxed{!}(x) \\
\boxed{!}(x) d' = d' x & x + y = y + x & x + 0 = x & (x+y)z = xz + yz \\
z(x+y) = zx + zy & (x+y)^* = x^* + y^* & !(x+y) = !x + !y & \boxed{!}(x+y) = \boxed{!}x + \boxed{!}y
\end{array}$$

▶ Remark 37. The equations in the definition above are mostly the same as the standard dynamic algebra Λ^* [10, 11] except those equations concerning the symbols with $'$ and the operator $\boxed{!}$, and their structures are analogous to those for $!$ operator. This again reflects the fact that the logical structure of rules for $\boxed{!}$ is analogous to that of $!$.

▶ **Definition 38.** *A* label *is an element of* $\Lambda^{\Box *}$ *that is associated to edges of proof structures as in Figure 16. Let* S *be a proof structure and* T_S *be the set of edge traversals in the structure.* S *is associated with a function* $w \colon T_S \to \Lambda^{\Box *}$ *defined by* $w(e) = l$ *(resp.* l^**) if* e *is a forward (resp. backward) traversal of an edge* e *and* l *is the label of the edge;* $w(e_1 e_2) = w(e_1) w(e_2)$*.*

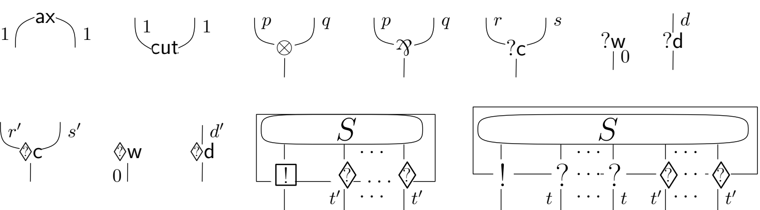


Figure 16 Labels on edges.

▶ **Definition 39.** *A* walk *over a proof structure S is an element of $\Lambda^{\Box *}$ that is obtained by concatenating labels along a graph-theoretic path over S such that the graph-theoretic path does not traverse an edge forward (resp. backward) immediately after the same edge backward (resp. forward); and does not traverse a premise of one of $\otimes, \parr, c$ node and another premise of the same node immediately after that. A* path *is a walk that is not proved to be equal to* 0*. A path is called* maximal *if it starts and finishes at a conclusion.*

The intuition of the notion of path is that a path is a "correct way" of traversing a proof net, in the sense that any path is preserved before and after a reduction. All the other walks that are not paths will be broken, which is represented by the constant 0 of $\Lambda^{\Box *}$. Then, we obtain a *context semantics* from paths in the following way.

▶ **Definition 40.** *Given a monomial path a, its action $[\![a]\!] : \Sigma \rightharpoonup \Sigma$ on contexts is defined as follows. We define $[\![1]\!]$ as the identity mapping on contexts. There is no definition of $[\![0]\!]$. The $[\![f^*]\!]$ is the inverse translation, i.e., $[\![f]\!]^{-1}$. The transformer of the composition of a and b is defined as $[\![ab]\!](m) \stackrel{\text{def}}{=} [\![a]\!]([\![b]\!](m))$. For the other labels, the interpretation are defined as follows where exponential morphisms ! and $\boxdot$ are defined by the meta-level pattern matchings:*

$$[\![p]\!](\mathcal{M},\mathcal{B},\mathcal{N}) \stackrel{\text{def}}{=} (l.\mathcal{M},\mathcal{B},\mathcal{N}) \qquad [\![q]\!](\mathcal{M},\mathcal{B},\mathcal{N}) \stackrel{\text{def}}{=} (r.\mathcal{M},\mathcal{B},\mathcal{N})$$

$$[\![r]\!](\mathcal{M},\mathcal{B},\mathcal{N}) \stackrel{\text{def}}{=} (\mathcal{M},L.\mathcal{B},\mathcal{N}) \qquad [\![s]\!](\mathcal{M},\mathcal{B},\mathcal{N}) \stackrel{\text{def}}{=} (\mathcal{M},R.\mathcal{B},\mathcal{N})$$

$$[\![t]\!](\mathcal{M},\langle\mathcal{B}_1,\langle\mathcal{B}_2,\mathcal{B}_3\rangle\rangle,\mathcal{N}) \stackrel{\text{def}}{=} (\mathcal{M},\langle\langle\mathcal{B}_1,\mathcal{B}_2\rangle,\mathcal{B}_3\rangle,\mathcal{N}) \qquad [\![d]\!](\mathcal{M},\mathcal{B},\mathcal{N}) \stackrel{\text{def}}{=} (\mathcal{M},\star.\mathcal{B},\mathcal{N})$$

$$[\![r']\!](\mathcal{M},\mathcal{B},\mathcal{N}) \stackrel{\text{def}}{=} (\mathcal{M},\mathcal{B},L'.\mathcal{N}) \qquad [\![s']\!](\mathcal{M},\mathcal{B},\mathcal{N}) \stackrel{\text{def}}{=} (\mathcal{M},\mathcal{B},R'.\mathcal{N})$$

$$[\![t']\!](\mathcal{M},\mathcal{B},\langle\mathcal{N}_1,\langle\mathcal{N}_2,\mathcal{N}_3\rangle\rangle) \stackrel{\text{def}}{=} (\mathcal{M},\mathcal{B},\langle\langle\mathcal{N}_1,\mathcal{N}_2\rangle,\mathcal{N}_3\rangle) \qquad [\![d']\!](\mathcal{M},\mathcal{B},\mathcal{N}) \stackrel{\text{def}}{=} (\mathcal{M},\mathcal{B},\star.\mathcal{N})$$

$$[\![!(f)]\!](\mathcal{M},\langle\mathcal{B}_1,\mathcal{B}_2\rangle,\mathcal{N}) \stackrel{\text{def}}{=} \textbf{\textit{let}}\ (\mathcal{M}',\mathcal{B}_2',\mathcal{N}') = [\![f]\!](\mathcal{M},\mathcal{B}_2,\mathcal{N})\ \textbf{\textit{in}}\ (\mathcal{M}',\langle\mathcal{B}_1,\mathcal{B}_2'\rangle,\mathcal{N}')$$

$$[\![\boxdot(f)]\!](\mathcal{M},\mathcal{B},\langle\mathcal{N}_1,\mathcal{N}_2\rangle) \stackrel{\text{def}}{=} \textbf{\textit{let}}\ (\mathcal{M}',\mathcal{B}',\mathcal{N}_2') = [\![f]\!](\mathcal{M},\mathcal{B},\mathcal{N}_2)\ \textbf{\textit{in}}\ (\mathcal{M}',\mathcal{B}',\langle\mathcal{N}_1,\mathcal{N}_2'\rangle)$$

Given a path a, its action $[\![a]\!] : \Sigma \rightharpoonup \mathbb{M}(\Sigma)$ is defined by the rules above (regarding the codomain as a multiset) and $[\![a+b]\!](m) = ([\![a]\!](m)) \uplus ([\![b]\!](m))$ where $\uplus$ is the multiset sum.

▶ Remark 41. In Mackie's work [10], the multiset in the codomain is not used since the main interest of his work is on terms of a base type: in that setting any proof net corresponding to a term has an execution formula that is monomial. In general, this style of context semantics is slightly degenerated compared to Girard's original version and its successors because the information of "current position" is dropped from the definition of contexts.

▶ **Definition 42.** *Let S be a closed proof net and χ be the set of maximal paths between conclusions of S. The* execution formula *is defined by $\mathcal{EX}(S) = \Sigma_{\phi\in\chi}\phi$ where the RHS is the sum of all paths in χ. The* context semantics *of S is defined to be $[\![\mathcal{EX}(S)]\!] : \Sigma \rightharpoonup \mathbb{M}(\Sigma)$.*

▶ **Definition 43.** *Let M be a closed well-typed term in $\lambda^{\boxdot}$. The* context semantics *of M is defined to be $[\![(M)^\dagger]\!]$, where $(-)^\dagger$ is a straightforward translation from $\lambda^{\boxdot}$-terms to proof nets, defined by constructing proof nets from $\lambda^{\boxdot}$-derivations as in Figure 17.*

▶ **Lemma 44.** *Let S be a closed proof net and S' be its normal form. Then $[\![S]\!] = [\![S']\!]$.*

The lemma is proved through two auxiliary lemmata below.

▶ **Lemma 45.** *Let ϕ be a path from a conclusion of a closed net S ending at a node a. Let $(\mathcal{M}',\mathcal{B}',\mathcal{N}') = [\![\phi]\!](\mathcal{M},\varepsilon,\varepsilon)$. The height of $\mathcal{B}'$ (resp. $\mathcal{N}'$) matches with the number of exponential (resp. necessitation) boxes containing the node a.*

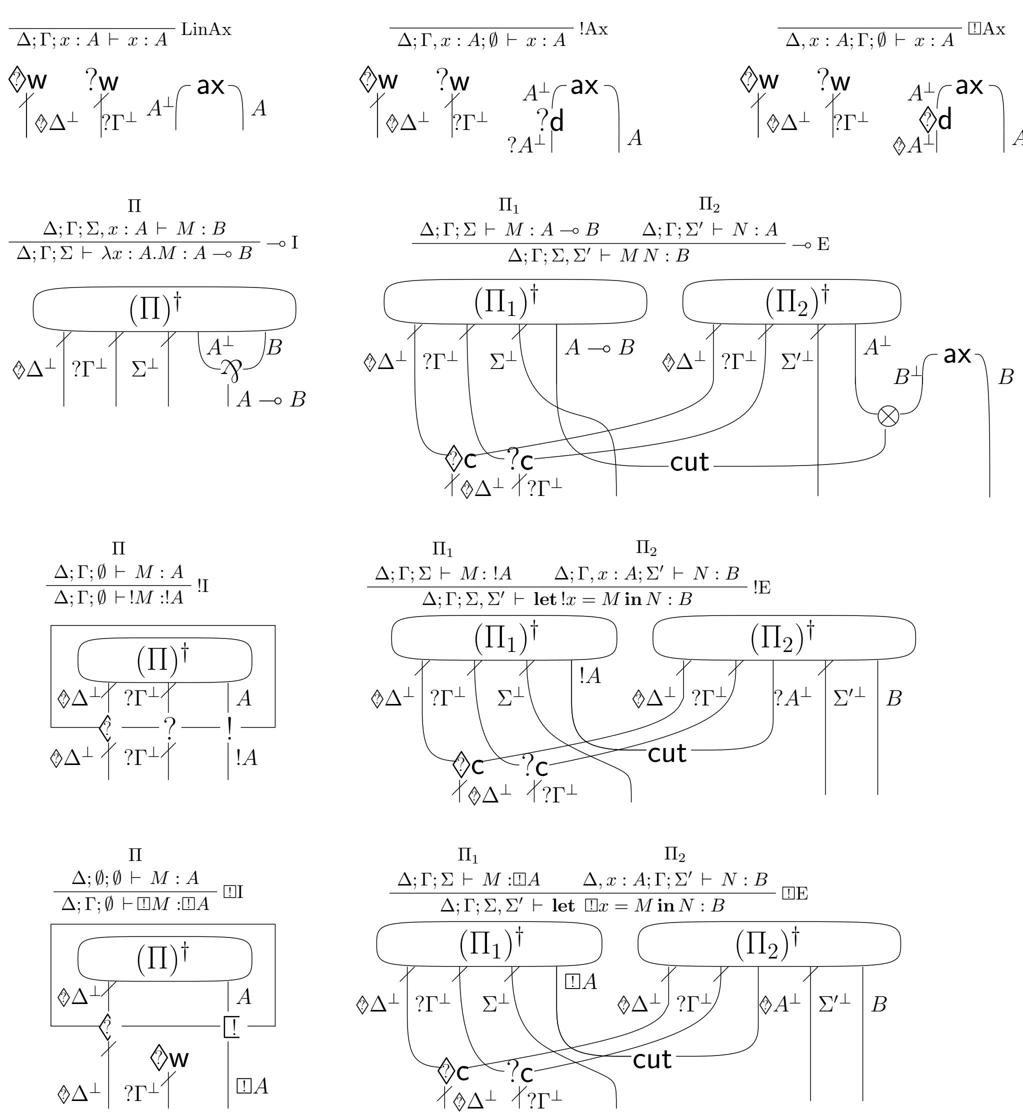


Figure 17 Translation from $\lambda^{\boxed{!}}$ to CMELL$^{\boxed{!}}$ proof nets.

Proof. By spectating the rules of actions above: the height of stacks only changes at doors of a box. ◀

▶ **Lemma 46.** *Let ϕ be a path inside a box of a closed net S. $[\![\phi]\!](\mathcal{M},\sigma.\mathcal{B},\tau.\mathcal{N})$ is in the form $(\mathcal{M}',\sigma'.\mathcal{B},\tau.\mathcal{N})$.*

Proof. Again, by spectating the rules of actions. ◀

▶ **Theorem 47.** *If a closed term M in $\lambda^{\boxed{!}}$ is typable and $M \rightsquigarrow M'$, then $[\![(M)^\dagger]\!] = [\![(M')^\dagger]\!]$.*

▶ Remark 48. This notion of context semantics inherently captures the "dynamics" of computation, and indeed Mackie exploited [10, 11] the character to implement a compiler, in the level of machine code, for PCF. In this paper we do not cover such a concrete compiler, but the definition of $[\![-]\!]$ can be seen as "context transformers" of virtual machine that is mathematically rigorous enough to model the computation of $\lambda^{\boxed{!}}$ (and hence of $\lambda^{\Box}$).

7 Related work

7.1 Linear-logical reconstruction of modal logic

The work on translations from modal logic to linear logic goes back to Martini and Masini [13]. They proposed a translation from classical S4 (CS4) to full propositional linear logic by means of the Grisin–Ono translation. However, their work only discusses provability.

The most similar work to ours is a "linear analysis" of CS4 by Schellinx [20], in which Girard translation from CS4 with respect to proofs is proposed. He uses a bi-colored linear logic, a subsystem of multicolored linear logic by Danos et al. [4], called **2-LL**, for a target calculus of the translation. It has two pairs of exponentials $\langle \underset{0}{!}, \underset{0}{?} \rangle$ and $\langle \underset{1}{!}, \underset{1}{?} \rangle$, called *subexponentials* following the terminology by Nigam and Miller [14], with the following rules:

$$\frac{\underset{1}{!}\Gamma, \underset{0}{!}\Gamma' \vdash A, \underset{1}{?}\Delta, \underset{0}{?}\Delta'}{\underset{1}{!}\Gamma, \underset{0}{!}\Gamma' \vdash \underset{0}{!}A, \underset{1}{?}\Delta, \underset{0}{?}\Delta'}\ \underset{0}{!}\mathrm{R} \qquad\qquad \frac{\underset{1}{!}\Gamma \vdash A, \underset{1}{?}\Delta}{\underset{1}{!}\Gamma \vdash \underset{1}{!}A, \underset{1}{?}\Delta}\ \underset{1}{!}\mathrm{R}$$

These rules have, while they are defined in the classical setting, essentially the same structure to what we defined as !R and $\boxed{!}$R for IMELL$^{\boxed{!}}$, respectively.

To mention the difference between the results of Schellinx and ours, his work has investigated only in terms of proof theory. Neither a typed λ-calculus nor a Geometry of Interaction interpretation was given. However, even so, he already gave a reduction-preserving Girard translation for the sequent calculi of CS4 and **2-LL**, and his *linear decoration* (cf. [20, 4]) allows us to obtain the cut-elimination theorem for CS4 as a corollary of that of **2-LL**. Thus, it should be interesting to investigate a relationship between his work and ours.

Furthermore, there also exist two uniform logical frameworks that can encode various logics including IS4 and CS4. One is the work by Nigam et al. [15] which based on Nigam and Miller's linear-logical framework with subexponentials and on the notion of focusing by Andreoli. The other work is *adjoint logic* by Pruiksma et al. [19] which based on, again, subexponentials, and the so-called LNL model for intuitionistic linear logic by Benton. While our present work is still far from the two works, it seems fruitful to take our discussion into their frameworks to give linear-logical computational interpretations for various logics.

7.2 Computation of modal logic and its relation to metaprogramming

Computational interpretations of modal logic have been considered not only for intuitionistic S4 but also for various logics, including the modal logics K, T, K4, and GL, and a few constructive temporal logics (cf. the survey by Kavvos in [8]). This field of modal logics is known to be connected to "metaprogramming" in the theory of programming languages and has been substantially studied. One of the studies is *(multi-)staged computation* (cf. [22]), which is a programming paradigm that supports Lisp-like *quasi-quote*, *unquote*, and *eval*. The work of $\lambda^{\Box}$ by Davies and Pfenning [5] is actually one of logical investigations of it.

Furthermore, the multi-stage programming is not a mere theory but has "real" implementations such as MetaML [22] and MetaOCaml (cf. a survey in [3]) in the style of functional programming languages. Some core calculi of these implementations are formalized as type systems (e.g. [21, 3]) and investigated from the logical point of view (e.g. [24]).

8 Conclusion

We have presented a linear-logical reconstruction of the intuitionistic modal logic S4, by establishing the modal linear logic with the $\boxed{!}$-modality and the S4-version of Girard

translation from IS4. The translation from IS4 to the modal linear logic is shown to be correct with respect to the level of proofs, through the Curry–Howard correspondence.

While the proof-level Girard translation for modal logic is already proposed by Schellinx, our typed λ-calculus $\lambda^{\boxed{!}}$ and its Geometry of Interaction Machine (GoIM) are novel. Also, the significance of our formalization is its simplicity. All we need to establish the linear-logical reconstruction of modal logic is the $\boxed{!}$-modality, an integration of !-modality and $\Box$-modality, that gives the structure of modal logic into linear logic. Thanks to the simplicity, our λ-calculus and the GoIM can be obtained as simple extensions of existing works.

As a further direction, we plan to enrich our framework to cover other modal logics such as K, T, and K4, following the work of contextual modal calculi by Kavvos [8]. Moreover, reinvestigating of the modal-logical foundation for multi-stage programming by Tsukada and Igarashi [24] via our methods and extending Mackie's GoIM for PCF [11] to the modal-logical setting seem to be interesting from the viewpoint of programming languages.

Lastly, we have also left a semantical study for modal linear logic with respect to the validity. At the present stage, we think that we could give a sound-and-complete characterization of modal linear logic by an integration of Kripke semantics of modal logic and phase semantics of linear logic, but details will be studied in a future paper.

References

1. Samson Abramsky, Esfandiar Haghverdi, and Philip Scott. Geometry of Interaction and Linear Combinatory Algebras. *Mathematical Structures in Computer Science*, 12(5):625–665, 2002. doi:10.1017/S0960129502003730.
2. Andrew Barber. Dual intuitionistic linear logic, 1996. Technical report LFCS-96-347. URL: http://www.lfcs.inf.ed.ac.uk/reports/96/ECS-LFCS-96-347.
3. Cristiano Calcagno, Eugenio Moggi, and Walid Taha. ML-like inference for classifiers. In *Proceedings of ESOP 2004*, pages 79–93, 2004. doi:10.1007/978-3-540-24725-8_7.
4. Vincent Danos, Jean-Baptiste Joinet, and Harold Schellinx. The Structure of Exponentials: Uncovering the Dynamics of Linear Logic Proofs. In G. Gottlob, A. Leitsch, and D. Mundici, editors, *Computational Logic and Proof Theory*, pages 159–171. Springer-Verlag, 1993. doi:10.1007/BFb0022564.
5. Rowan Davies and Frank Pfenning. A Modal Analysis of Staged Computation. *Journal of the ACM*, 48(3):555–604, 2001. doi:10.1145/382780.382785.
6. Yosuke Fukuda and Akira Yoshimizu. A Higher-arity Sequent Calculus for Modal Linear Logic. In *RIMS Kôkyûroku 2083: Symposium on Proof Theory and Proving*, pages 76–87, 2018. URL: http://www.kurims.kyoto-u.ac.jp/~kyodo/kokyuroku/contents/pdf/2083-07.pdf.
7. Jean-Yves Girard. Linear Logic. *Theoretical Compututer Science*, 50(1):1–102, 1987. doi:10.1016/0304-3975(87)90045-4.
8. G. A. Kavvos. Dual-Context Calculi for Modal Logic. In *Proceedings of LICS 2017*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005089.
9. Patrick Lincoln, John Mitchell, Andre Scedrov, and Natarajan Shankar. Decision problems for propositional linear logic. *Annals of Pure and Applied Logic*, 56(1):239–311, 1992. doi:10.1016/0168-0072(92)90075-B.
10. Ian Mackie. *The Geometry of Implementation.* PhD thesis, University of London, 1994.
11. Ian Mackie. The Geometry of Interaction Machine. In *Proceedings of POPL 1995*, pages 198–208, 1995. doi:10.1145/199448.199483.
12. John Maraist, Martin Odersky, David N. Turner, and Philip Wadler. Call-by-name, call-by-value, call-by-need and the linear lambda lalculus. *Theoretical Computer Science*, 228(1-2):175–210, 1999. doi:10.1016/S0304-3975(98)00358-2.
13. Simone Martini and Andrea Masini. A Modal View of Linear Logic. *Journal of Symbolic Logic*, 59(3):888–899, 1994. doi:10.2307/2275915.

14 Vivek Nigam and Dale Miller. Algorithmic Specifications in Linear Logic with Subexponentials. In *Proceedings of PPDP 2009*, pages 129–140, 2009. doi:10.1145/1599410.1599427.
15 Vivek Nigam, Elaine Pimentel, and Giselle Reis. An extended framework for specifying and reasoning about proof systems. *Journal of Logic and Computation*, 26(2):539–576, 2016. doi:10.1093/logcom/exu029.
16 Yo Ohta and Masahito Hasegawa. A Terminating and Confluent Linear Lambda Calculus. In *Proceedings of RTA 2006*, pages 166–180, 2006. doi:10.1007/11805618_13.
17 Frank Pfenning. Lecture Notes on Combinatory Modal Logic, 2010. Lecture note. URL: http://www.cs.cmu.edu/~fp/courses/15816-s10/lectures/09-combinators.pdf.
18 Frank Pfenning and Rowan Davies. A judgmental reconstruction of modal logic. *Mathematical Structures in Computer Science*, 11(4):511–540, 2001. doi:10.1017/S0960129501003322.
19 Klaas Pruiksma, William Chargin, Frank Pfenning, and Jason Reed. Adjoint Logic, 2018. Manuscript. URL: http://www.cs.cmu.edu/~fp/papers/adjoint18b.pdf.
20 Harold Schellinx. A Linear Approach to Modal Proof Theory. In *Proof Theory of Modal Logic*, pages 33–43. Springer, 1996. doi:10.1007/978-94-017-2798-3_3.
21 Walid Taha and Michael Florentin Nielsen. Environment Classifiers. In *Proceedings of POPL 2003*, pages 26–37, 2003. doi:10.1145/640128.604134.
22 Walid Taha and Tim Sheard. MetaML and multi-stage programming with explicit annotations. *Theoretical Computer Science*, 248(1-2):211–242, 2000. doi:10.1016/S0304-3975(00)00053-0.
23 A. S. Troelstra and H. Schwichtenberg. *Basic Proof Theory*. Cambridge University Press, 1996.
24 Takeshi Tsukada and Atsushi Igarashi. A Logical Foundation for Environment Classifiers. *Logical Methods in Computer Science*, 6(4:8):1–43, 2010. doi:10.2168/LMCS-6(4:8)2010.

A Appendix

A.1 Cut-elimination theorem for the intuitionistic modal linear logic

In this section, we give a complete proof of the cut-elimination theorem of $\mathrm{IMELL}^{\boxdot}$.

▶ **Theorem 49** (Cut-elimination). *The rule* Cut *in* $\mathrm{IMELL}^{\boxdot}$ *is admissible, i.e., if* $\Gamma \vdash A$ *is derivable, then there is a derivation of the same judgment without any applications of* Cut.

Proof. As we mentioned in the body, we will show the following rules are admissible.

$$\frac{\Gamma \vdash !A \qquad \Gamma', (!A)^n \vdash B}{\Gamma, \Gamma' \vdash B}\ !\mathrm{Cut} \qquad\qquad \frac{\Gamma \vdash \boxdot A \qquad \Gamma', (\boxdot A)^n \vdash B}{\Gamma, \Gamma' \vdash B}\ \boxdot\mathrm{Cut}$$

The admissibility of Cut, !Cut, $\boxdot$Cut are shown by simultaneous induction on the derivation of $\Gamma \vdash A$ with the lexicographic complexity $\langle \delta, h \rangle$, where δ is the degree of the assumed derivation and h is its height. Therefore, it is enough to show that for every application of cuts, one of the following hold: (1) it can be reduced to a cut with a smaller cut-degree; (2) it can be reduced to a cut with a smaller height; (3) it can be eliminated immediately.

In what follows, we will explain the admissibility of each cut rule separately although the actual proofs are done simultaneously.

- The admissibility of Cut. We show that every application of the rule Cut whose cut-degree is maximal is eliminable. Thus, consider an application of Cut in the derivation:

$$\frac{\begin{array}{c}\Pi_0\\ \Gamma \vdash A\end{array} \qquad \begin{array}{c}\Pi_1\\ \Gamma', A \vdash B\end{array}}{\Gamma, \Gamma' \vdash B}\ \mathrm{Cut}$$

  such that its cut-degree is maximal and its height is minimal (comparing to the other applications whose cut-degree is maximal). The proof proceeds by case analysis on Π_0.

- Π_0 ends with Cut. In this case the derivation is as follows:

$$\dfrac{\dfrac{\begin{matrix}\vdots \\ \Gamma_0 \vdash C\end{matrix} \qquad \begin{matrix}\vdots \\ \Gamma_1, C \vdash A\end{matrix}}{\Gamma_0, \Gamma_1 \vdash A}\,\text{Cut} \qquad \begin{matrix}\Pi_1 \\ \Gamma', A \vdash B\end{matrix}}{\Gamma_0, \Gamma_1, \Gamma' \vdash B}\,\text{Cut}$$

  Since the bottom application of Cut was chosen to have the maximal cut-degree and the minimum height, the cut-degree of the above is less than that of the bottom. Therefore, the derivation can be translated to the following:

$$\dfrac{\dfrac{\begin{matrix}\vdots \\ \Gamma_0 \vdash C\end{matrix} \qquad \dfrac{\dfrac{\begin{matrix}\vdots \\ \Gamma_1, C \vdash A\end{matrix} \qquad \begin{matrix}\Pi_1 \\ \Gamma', A \vdash B\end{matrix}}{\Gamma_1, C, \Gamma' \vdash B}\,\text{I.H.}}{}}{\Gamma_0, \Gamma_1, \Gamma' \vdash B}\,\text{I.H.}$$

- Π_0 ends with $\multimap$R. In this case, the derivation is as follows:

$$\dfrac{\dfrac{\begin{matrix}\vdots \\ \Gamma, A_0 \vdash A_1\end{matrix}}{\Gamma \vdash A_0 \multimap A_1}\,\multimap\text{R} \qquad \begin{matrix}\Pi_1 \\ \Gamma', A_0 \multimap A_1 \vdash B\end{matrix}}{\Gamma, \Gamma' \vdash B}\,\text{Cut}$$

  for some A_0 and A_1 such that $A \equiv A_0 \multimap A_1$. If the last step in Π_1 is Ax, then the result is obtained as Π_0. If the last step in Π_1 is $\multimap$L, the derivation is as follows:

$$\dfrac{\dfrac{\begin{matrix}\vdots \\ \Gamma, A_0 \vdash A_1\end{matrix}}{\Gamma \vdash A_0 \multimap A_1}\,\multimap\text{R} \qquad \dfrac{\begin{matrix}\vdots \\ \Gamma' \vdash A_0\end{matrix} \qquad \begin{matrix}\vdots \\ \Gamma'', A_1 \vdash B\end{matrix}}{\Gamma', \Gamma'', A_0 \multimap A_1 \vdash B}\,\multimap\text{L}}{\Gamma, \Gamma', \Gamma'' \vdash B}\,\text{Cut}$$

  which is translated to the following:

$$\dfrac{\dfrac{\begin{matrix}\vdots \\ \Gamma' \vdash A_0\end{matrix} \qquad \begin{matrix}\vdots \\ \Gamma, A_0 \vdash A_1\end{matrix}}{\Gamma, \Gamma' \vdash A_1}\,\text{I.H.} \qquad \begin{matrix}\vdots \\ \Gamma'', A_1 \vdash B\end{matrix}}{\Gamma, \Gamma', \Gamma'' \vdash B}\,\text{I.H.}$$

  since the cut-degrees of A_0 and A_1 are less than that of A. The other cases can be shown by simple commutative conversions.
- Π_0 ends with !R. This case is dealt as a special case of the case !R in !Cut.
- Π_0 ends with $\boxed{!}$R. This case is dealt as a special case of the case $\boxed{!}$R in $\boxed{!}$Cut.
- Π_0 ends with the other rules. Easy.

- The admissibility of !Cut. As in the case of Cut, consider an application of !Cut:

$$\dfrac{\begin{matrix}\Pi_0 \\ \Gamma \vdash !A\end{matrix} \qquad \begin{matrix}\Pi_1 \\ \Gamma', (!A)^n \vdash B\end{matrix}}{\Gamma, \Gamma' \vdash B}\,!\text{Cut}$$

  such that its cut-degree is maximal and its height is minimal. By case analysis on Π_0.
  - Π_0 ends with Ax. In this case the cut-elimination is done as follows:

$$\dfrac{\dfrac{}{!A \vdash !A} \qquad \begin{matrix}\Pi_1 \\ \Gamma', (!A)^n \vdash B\end{matrix}}{!A, \Gamma' \vdash B}\,!\text{Cut} \quad \overset{\text{Cut elim.}}{\Longrightarrow} \quad \dfrac{\begin{matrix}\Pi_1 \\ \Gamma', (!A)^n \vdash B\end{matrix}}{\Gamma', !A \vdash B}\,!\text{C}$$

- Π_0 ends with !R. In this case the derivation is as follows:

$$\dfrac{\dfrac{\begin{matrix}\vdots\\ \boxed{!}\Gamma_0, !\Gamma_1 \vdash A\end{matrix}}{\boxed{!}\Gamma_0, !\Gamma_1 \vdash !A}\ !\mathrm{R} \qquad \begin{matrix}\Pi_1\\ \Gamma', (!A)^n \vdash B\end{matrix}}{\boxed{!}\Gamma_0, !\Gamma_1, \Gamma' \vdash B}\ !\mathrm{Cut}$$

Due to the side-condition of !R, we have to do case analysis on Π_1 further as follows.

  - Π_1 ends with Cut. In this case the derivation is as follows:

$$\dfrac{\begin{matrix}\Pi_0\\ \boxed{!}\Gamma_0, !\Gamma_1 \vdash !A\end{matrix} \qquad \dfrac{\begin{matrix}\vdots\\ \Gamma', (!A)^k \vdash C\end{matrix} \qquad \begin{matrix}\vdots\\ \Gamma'', (!A)^l, C \vdash B\end{matrix}}{\Gamma', \Gamma'', (!A)^n \vdash B}\ \mathrm{Cut}}{\boxed{!}\Gamma_0, !\Gamma_1, \Gamma', \Gamma'' \vdash B}\ !\mathrm{Cut}$$

  where $n = k + l$. We only deal with the case of $k > 0$ and $l > 0$, and the other cases are easy. Then, the derivation can be translated to the following:

$$\dfrac{\dfrac{\dfrac{\begin{matrix}\Pi_0\\ \boxed{!}\Gamma_0, !\Gamma_1 \vdash !A\end{matrix} \qquad \begin{matrix}\vdots\\ \Gamma', (!A)^k \vdash C\end{matrix}}{\boxed{!}\Gamma_0, !\Gamma_1, \Gamma' \vdash C}\ \mathrm{I.H.} \qquad \dfrac{\begin{matrix}\Pi_0\\ \boxed{!}\Gamma_0, !\Gamma_1 \vdash !A\end{matrix} \qquad \begin{matrix}\vdots\\ \Gamma'', (!A)^l, C \vdash B\end{matrix}}{\boxed{!}\Gamma_0, !\Gamma_1, \Gamma'', C \vdash B}\ \mathrm{I.H.}}{(\boxed{!}\Gamma_0)^2, (!\Gamma_1)^2, \Gamma', \Gamma'' \vdash B}\ \mathrm{I.H.}}{\boxed{!}\Gamma_0, !\Gamma_1, \Gamma', \Gamma'' \vdash B}\ !\mathrm{C}, \boxed{!}\mathrm{C}$$

  since the cut-degree of !Cut is less than that of Cut from the assumption.

  - Π_1 ends with !L. If the formula introduced by !L is not the cut-formula, then it is easy. For the other case, the derivation is as follows:

$$\dfrac{\dfrac{\begin{matrix}\vdots\\ \boxed{!}\Gamma_0, !\Gamma_1 \vdash A\end{matrix}}{\boxed{!}\Gamma_0, !\Gamma_1 \vdash !A}\ !\mathrm{R} \qquad \dfrac{\begin{matrix}\vdots\\ \Gamma', (!A)^{n-1}, A \vdash B\end{matrix}}{\Gamma', (!A)^n \vdash B}\ !\mathrm{L}}{\boxed{!}\Gamma_0, !\Gamma_1, \Gamma' \vdash B}\ !\mathrm{Cut}$$

  which is translated to the following:

$$\dfrac{\dfrac{\begin{matrix}\vdots\\ \boxed{!}\Gamma_0, !\Gamma_1 \vdash A\end{matrix} \qquad \dfrac{\begin{matrix}\Pi_0\\ \boxed{!}\Gamma_0, !\Gamma_1 \vdash !A\end{matrix} \qquad \begin{matrix}\vdots\\ \Gamma', (!A)^{n-1}, A \vdash B\end{matrix}}{\boxed{!}\Gamma_0, !\Gamma_1, \Gamma', A \vdash B}\ \mathrm{I.H.}}{(\boxed{!}\Gamma_0)^2, (!\Gamma_1)^2, \Gamma' \vdash B}\ \mathrm{I.H.}}{\boxed{!}\Gamma_0, !\Gamma_1, \Gamma' \vdash B}\ !\mathrm{C}, \boxed{!}\mathrm{C}$$

  - Π_1 ends with !C. If the formula introduced by !C is not the cut-formula, then it is easy. For the other case, the cut-elimination is done as follows:

$$\dfrac{\begin{matrix}\Pi_0\\ \boxed{!}\Gamma_0, !\Gamma_1 \vdash !A\end{matrix} \qquad \dfrac{\begin{matrix}\vdots\\ \Gamma', (!A)^{n+1} \vdash B\end{matrix}}{\Gamma', (!A)^n \vdash B}\ !\mathrm{C}}{\boxed{!}\Gamma_0, !\Gamma_1, \Gamma' \vdash B}\ !\mathrm{Cut} \quad \overset{\text{Cut elim.}}{\Longrightarrow} \quad \dfrac{\begin{matrix}\Pi_0\\ \boxed{!}\Gamma_0, !\Gamma_1 \vdash !A\end{matrix} \quad \begin{matrix}\vdots\\ \Gamma', (!A)^{n+1} \vdash B\end{matrix}}{\boxed{!}\Gamma_0, !\Gamma_1, \Gamma' \vdash B}\ \mathrm{I.H.}$$

  Note that the whole proof has not been proceeding by induction on n, and hence the number of occurrences of $!A$ does not matter in this case.

  - Π_1 ends with the other rules. Easy.

- Π_0 ends with the other rules. Easy.

- The admissibility of $\boxed{!}$Cut. Similar to the case of !Cut. ◀

Syntactic category

Types $A, B, C ::= p \mid A \multimap B \mid !A$

Terms $M, N, L ::= x \mid \lambda x : A.M \mid M\,N \mid !M \mid \mathbf{let}\,!x = M\,\mathbf{in}\,N$

Typing rule

$$\frac{}{\Gamma; x : A \vdash x : A}\text{LinAx} \qquad \frac{}{\Gamma, x : A; \emptyset \vdash x : A}\text{!Ax}$$

$$\frac{\Gamma; \Sigma, x : A \vdash M : B}{\Gamma; \Sigma \vdash \lambda x : A.M : A \multimap B}\multimap\text{I} \qquad \frac{\Gamma; \Sigma \vdash M : A \multimap B \qquad \Gamma; \Sigma' \vdash N : A}{\Gamma; \Sigma, \Sigma' \vdash M\,N : B}\multimap\text{E}$$

$$\frac{\Gamma; \emptyset \vdash M : A}{\Gamma; \emptyset \vdash !M : !A}\text{!I} \qquad \frac{\Gamma; \Sigma \vdash M : !A \qquad \Gamma, x : A; \Sigma' \vdash N : B}{\Gamma; \Sigma, \Sigma' \vdash \mathbf{let}\,!x = M\,\mathbf{in}\,N : B}\text{!E}$$

Reduction rule

$$(\lambda x : A.M)\,N \rightsquigarrow M[x := N]$$
$$\lambda x : A.M\,x \rightsquigarrow M$$
$$\mathbf{let}\,!x = !M\,\mathbf{in}\,N \rightsquigarrow N[x := M]$$
$$\mathbf{let}\,!x = M\,\mathbf{in}\,C[!x] \rightsquigarrow C[M]$$
$$(\mathbf{let}\,!x = M\,\mathbf{in}\,N)\,L \rightsquigarrow \mathbf{let}\,!x = M\,\mathbf{in}\,N\,L$$
$$\mathbf{let}\,!y = (\mathbf{let}\,!x = M\,\mathbf{in}\,N)\,\mathbf{in}\,L \rightsquigarrow \mathbf{let}\,!x = M\,\mathbf{in}\,\mathbf{let}\,!y = N\,\mathbf{in}\,L$$
$$\lambda y : A.(\mathbf{let}\,!x = M\,\mathbf{in}\,N) \rightsquigarrow \mathbf{let}\,!x = M\,\mathbf{in}\,\lambda y : A.N \qquad (\text{if } y \notin \mathrm{FV}(M))$$
$$L\,(\mathbf{let}\,!x = M\,\mathbf{in}\,N) \rightsquigarrow \mathbf{let}\,!x = M\,\mathbf{in}\,L\,N$$
$$\lambda z : A.(\mathbf{let}\,!y = M\,\mathbf{in}\,\mathbf{let}\,!x = L\,\mathbf{in}\,N) \rightsquigarrow \mathbf{let}\,!x = L\,\mathbf{in}\,\lambda z : A.(\mathbf{let}\,!y = M\,\mathbf{in}\,N) \quad (\text{if } y \notin \mathrm{FV}(L))$$

where $C[-]$ is a linear context defined by the following grammar:

$$C ::= [-] \mid \lambda x : A.C \mid C\,M \mid M\,C \mid \mathbf{let}\,!x = C\,\mathbf{in}\,M \mid \mathbf{let}\,!x = M\,\mathbf{in}\,C$$

Figure 18 Definition of $\lambda^{!,\multimap}$ (some syntax are changed to fit the present paper's notation).

A.2 Strong normalizability of the typed λ-calculus for modal linear logic

We complete the proof of the strong normalization theorem for $\lambda^{\boxed{!}}$. As we mentioned, this is done by an embedding to a typed λ-calculus for the $(!, \multimap)$-fragment of dual intuitionistic linear logic, studied by Ohta and Hasegawa [16], and shown to be strongly normalizing.

The calculus of Ohta and Hasegawa, named $\lambda^{!,\multimap}$ here, is given in Figure 18. The syntax and the typing rules can be read in the same way as (the $(!, \multimap)$-fragment of) $\lambda^{\boxed{!}}$. There are somewhat many reduction rules in contrast to those of $\lambda^{\boxed{!}}$, but these are due to the purpose of Ohta and Hasegawa to consider η-rules and commutative conversions. The different sets of reduction rules do not cause any problems to prove the strong normalizability of $\lambda^{\boxed{!}}$.

▶ **Definition 50** (Embedding). *An* embedding *from* $\lambda^{\boxed{!}}$ *to* $\lambda^{!,\multimap}$ *is defined to be the triple of the translations* $(A)^{\ddagger}$, $(\Gamma)^{\ddagger}$, *and* $(M)^{\ddagger}$ *given in Figure 19.*

▶ **Lemma 51** (Preservation of typing and reduction).

1. *If* $\Delta; \Gamma; \Sigma \vdash M : A$ *in* $\lambda^{\boxed{!}}$, *then* $(\Delta, \Gamma)^{\ddagger}; (\Sigma)^{\ddagger} \vdash (M)^{\ddagger} : (A)^{\ddagger}$ *in* $\lambda^{!,\multimap}$.
2. *If* $M \rightsquigarrow N$ *in* $\lambda^{\boxed{!}}$, *then* $(M)^{\ddagger} \rightsquigarrow (N)^{\ddagger}$ *in* $\lambda^{!,\multimap}$.

Proof. By induction on $\Delta; \Gamma; \Sigma \vdash M : A$ and $M \rightsquigarrow N$, respectively. ◀

$\underline{(A)^\ddagger}$

$$(p)^\ddagger \stackrel{\text{def}}{=} p$$
$$(A \multimap B)^\ddagger \stackrel{\text{def}}{=} (A)^\ddagger \multimap (B)^\ddagger$$
$$(!A)^\ddagger \stackrel{\text{def}}{=} !(A)^\ddagger$$
$$(\Box A)^\ddagger \stackrel{\text{def}}{=} !(A)^\ddagger$$

$\underline{(\Gamma)^\ddagger}$

$$(\Gamma)^\ddagger \stackrel{\text{def}}{=} \{(x : (A)^\ddagger) \mid (x : A) \in \Gamma\}$$

$\underline{(M)^\ddagger}$

$$(x)^\ddagger \stackrel{\text{def}}{=} x$$
$$(\lambda x : A.M)^\ddagger \stackrel{\text{def}}{=} \lambda x : (A)^\ddagger.(M)^\ddagger$$
$$(M\ N)^\ddagger \stackrel{\text{def}}{=} (M)^\ddagger\,(N)^\ddagger$$
$$(!M)^\ddagger \stackrel{\text{def}}{=} !(M)^\ddagger$$
$$(\mathbf{let}\ !x = M\ \mathbf{in}\ N)^\ddagger \stackrel{\text{def}}{=} \mathbf{let}\ !x = (M)^\ddagger\ \mathbf{in}\ (N)^\ddagger$$
$$(\Box M)^\ddagger \stackrel{\text{def}}{=} !(M)^\ddagger$$
$$(\mathbf{let}\ \Box x = M\ \mathbf{in}\ N)^\ddagger \stackrel{\text{def}}{=} \mathbf{let}\ !x = (M)^\ddagger\ \mathbf{in}\ (N)^\ddagger$$

Figure 19 Definition of the embeddings $(A)^\ddagger$, $(\Gamma)^\ddagger$, and $(M)^\ddagger$.

$\underline{\lambda_* x.M}$

$$\lambda_* x.x \stackrel{\text{def}}{=} \mathrm{I}$$
$$\lambda_* x.(M\ N) \stackrel{\text{def}}{=} \mathrm{C}\,(\lambda_* x.M)\,N \quad \text{if } x \in \mathrm{FV}(M)$$
$$\lambda_* x.(M\ N) \stackrel{\text{def}}{=} \mathrm{B}\,M\,(\lambda_* x.N) \quad \text{if } x \in \mathrm{FV}(N)$$

$\underline{\lambda^!_* x.M}$

$$\lambda^!_* x.x \stackrel{\text{def}}{=} \mathrm{T}^!$$
$$\lambda^!_* x.M \stackrel{\text{def}}{=} \mathrm{K}^!\,M \quad \text{if } (a)$$
$$\lambda^!_* x.(M\ N) \stackrel{\text{def}}{=} \mathrm{C}\,(\lambda^!_* x.M)\,N \quad \text{if } (b)$$
$$\lambda^!_* x.(M\ N) \stackrel{\text{def}}{=} \mathrm{B}\,M\,(\lambda^!_* x.N) \quad \text{if } (c)$$
$$\lambda^!_* x.(M\ N) \stackrel{\text{def}}{=} \mathrm{S}^!\,(\lambda^!_* x.M)\,(\lambda^!_* x.N) \quad \text{if } (d)$$
$$\lambda^!_* x.(!M) \stackrel{\text{def}}{=} \mathrm{B}\,(\mathrm{D}^!\,!(\lambda^!_* x.M))\,4^!$$

$\underline{\lambda^\Box_* x.M}$

$$\lambda^\Box_* x.x \stackrel{\text{def}}{=} \mathrm{T}^\Box$$
$$\lambda^\Box_* x.M \stackrel{\text{def}}{=} \mathrm{K}^\Box M \quad \text{if } (a)$$
$$\lambda^\Box_* x.(M\ N) \stackrel{\text{def}}{=} \mathrm{C}\,(\lambda^\Box_* x.M)\,N \quad \text{if } (b)$$
$$\lambda^\Box_* x.(M\ N) \stackrel{\text{def}}{=} \mathrm{B}\,M\,(\lambda^\Box_* x.N) \quad \text{if } (c)$$
$$\lambda^\Box_* x.(M\ N) \stackrel{\text{def}}{=} \mathrm{S}^\Box(\lambda^\Box_* x.M)\,(\lambda^\Box_* x.N) \quad \text{if } (d)$$
$$\lambda^\Box_* x.(!M) \stackrel{\text{def}}{=} \mathrm{B}\,(\mathrm{D}^!\,(!(\lambda^\Box_* x.M)))\,(\mathrm{B}\,\mathrm{E}\,4^\Box)$$
$$\lambda^\Box_* x.(\Box M) \stackrel{\text{def}}{=} \mathrm{B}\,(\mathrm{D}^\Box(\Box(\lambda^\Box_* x.M)))\,4^\Box$$

where $(a), (b), (c), (d)$ means the conditions $(x \notin \mathrm{FV}(M))$, $(x \in \mathrm{FV}(M)$ and $x \notin \mathrm{FV}(N))$, $(x \notin \mathrm{FV}(M)$ and $x \in \mathrm{FV}(N))$, $(x \in \mathrm{FV}(M)$ and $x \in \mathrm{FV}(N))$, respectively.

Figure 20 Definitions of $(\lambda_* x.M)$, $(\lambda^!_* x.M)$, and $(\lambda^\Box_* x.M)$ for bracket abstraction.

▶ **Theorem 52** (Strong normalization)**.** *In* $\lambda^\Box$*, there are no infinite reduction sequences starting from* M *for all well-typed term* M*.*

Proof. Suppose that there exists an infinite reduction sequence starting from M in $\lambda^\Box$. Then, the term $(M)^\ddagger$ is well-typed in $\lambda^{!,\multimap}$ and yields an infinite reduction sequence in $\lambda^{!,\multimap}$ by Lemma 51. However, this contradicts the strong normalizability of $\lambda^{!,\multimap}$. ◀

A.3 Bracket abstraction algorithm

We show the definition of bracket abstraction operators in this section.

▶ **Definition 53** (Bracket abstraction)**.** *Let* M *be a term* M *of* $\mathrm{CL}^\Box$ *such that* $\Delta; \Gamma; \Sigma \vdash M : A$ *and* $x \in \mathrm{FV}(M)$ *for some* $\Delta, \Gamma, \Sigma, A$ *and* x*. Then, the* bracket abstraction of M with respect

to x *is defined to be either one of the following, depending the variable kind of* x*:*

$$\begin{array}{ll} (\lambda_* x.M) & \textit{if } x \in \mathrm{dom}(\Sigma)^3; \\ (\lambda^!_* x.M) & \textit{if } x \in \mathrm{dom}(\Gamma); \\ (\lambda^{\boxed{!}}_* x.M) & \textit{if } x \in \mathrm{dom}(\Delta), \end{array}$$

where each one of $(\lambda_* x.M)$*,* $(\lambda^!_* x.M)$*, and* $(\lambda^{\boxed{!}}_* x.M)$ *is the meta-level* bracket abstraction *operation given in Figure 20, which takes the pair of* x *and* M*, and yields a* $\mathrm{CL}^{\boxed{!}}$*-term.*

▶ Remark 54. As in the case of standard bracket abstraction algorithm, the intuition behind the operations $(\lambda_* x.M)$, $(\lambda^!_* x.M)$, and $(\lambda^!_* x.M)$ is that they are defined so as to mimic the λ-abstraction operation in the framework of combinatory logic. For instance, the denotation of $(\lambda_* x.M)$ is a $\mathrm{CL}^{\boxed{!}}$-term that represents a function with the parameter x, that is, it is a term that satisfies that $(\lambda_* x.M)\, N \rightsquigarrow^+ M[x := N]$ in $\mathrm{CL}^{\boxed{!}}$, for all $\mathrm{CL}^{\boxed{!}}$-terms N.

▶ Remark 55. There are no definitions for some cases in $(\lambda_* x.M)$ and $(\lambda^!_* x.M)$, e.g, the case that $(\lambda_* x.(M\, N))$ such that $x \in \mathrm{FV}(M)$ and $x \in \mathrm{FV}(N)$, and the case that $(\lambda^!_* x.(\boxed{!} M))$. This is because that these are actually unnecessary due to the linearity condition and the side condition of the rule $\boxed{!}$. Moreover, the well-definedness of the bracket abstraction operations can be shown by induction on M, and in reality, the proof of the deduction theorem can be seen as what justifies it. The intentions that $(\lambda_* x.M)\, N \rightsquigarrow^+ M[x := N]$, etc. can also be shown by easy calculation.

[3] $\mathrm{dom}(\Gamma)$ is defined to be the set $\{x \mid (x : A) \in \Gamma\}$ for all type contexts Γ.

Sparse Tiling Through Overlap Closures for Termination of String Rewriting

Alfons Geser
HTWK Leipzig, Germany

Dieter Hofbauer
ASW – Berufsakademie Saarland, Germany

Johannes Waldmann
HTWK Leipzig, Germany

Abstract

A strictly locally testable language is characterized by its set of admissible factors, prefixes and suffixes, called tiles. We over-approximate reachability sets in string rewriting by languages defined by sparse sets of tiles, containing only those that are reachable in derivations. Using the partial algebra defined by a tiling for semantic labeling, we obtain a transformational method for proving local termination. These algebras can be represented efficiently as finite automata of a certain shape. Using a known result on forward closures, and a new characterisation of overlap closures, we can automatically prove termination and relative termination, respectively. We report on experiments showing the strength of the method.

2012 ACM Subject Classification Theory of computation → Rewrite systems

Keywords and phrases relative termination, semantic labeling, locally testable language, overlap closure

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.21

1 Introduction

Methods for proving termination of rewriting (automatically) can be classified [25] into syntactical (using a precedence on letters), semantical (map each letter to a function on some domain), or transformational. Applying a transformation, one hopes to obtain an equivalent termination problem that is easier to handle.

One such transformation is *semantic labeling* [24]. This will typically increase the number of rules, sometimes drastically so. We consider here a specific semantic domain, called the *k-shift algebra*, consisting of words of length $k-1$, with the "shift left" operation.

When we use this algebra (in Section 3) for semantically labeling a string w, each labeled letter is a k-factor of w, called a *tile*.

Our implementation uses values of k from 2 to 8. *Self labeling* [17] can be seen as unrestricted shifting. The 2-shift algebra is used in *root labeling* [19] which first appeared in Termination Competitions in 2006. MultumNonMulta [12] took first place in both categories Standard and Relative SRS of the 2018 competition [13] mainly due to the use of 2-tiling.

A *sparse* tiling contains just those tiles that can occur during derivations starting in a given language. The shift algebra given by those tiles then is a *partial model* [3]. In Section 4, we present an algorithm to complete a given set of tiles with respect to a given rewriting system (i. e., to construct a minimal partial model) and provide an efficient implementation that can handle large sets of tiles.

We apply this method to derivations starting from right-hand sides of forwards closures (Section 5) and overlap closures (Section 7) since local termination on these languages implies

© Alfons Geser, Dieter Hofbauer, and Johannes Waldmann;
licensed under Creative Commons License CC-BY
4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019).
Editor: Herman Geuvers; Article No. 21; pp. 21:1–21:21
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

global termination (a known result), and relative termination (Section 6), respectively. In all, we obtain a transformational method for proving termination and relative termination: construct a closed set of tiles, and then use it for semantic labeling. This can be combined with other methods for proving termination, e. g., weights (linear interpretations of slope 1).

We obtain yet another automated termination proof for Zantema's Problem $\{a^2b^2 \to b^3a^3\}$, which is a classical benchmark, see Example 5.6. Our implementation is part of the Matchbox termination prover, and it easily solves several termination problems from the Termination Problems Database[1] that appear hard for other approaches, e. g., Examples 8.3 and 8.4. Sparse tiling contributed to Matchbox winning the categories *SRS Standard* and *SRS Relative* of the Termination Competition 2019, see Section 9.

Our application area is string rewriting, and our implementation is tailored to that. Still, for proving correctness, we use the language of term rewriting, as this allows to re-use concepts and results.

2 Notation

Given a set of *letters* Σ, i. e., an *alphabet*, a string is a finite sequence of letters over Σ. The number of its components is the *length* of the string, and the string of length zero, the *empty string*, is denoted by ϵ. If there is no ambiguity, we denote the string with letters $a_1, \dots, a_n$ by $a_1 \dots a_n$. We deal, however, also with strings of strings, and then use the list notation $[a_1, \dots, a_n]$. Let $\mathsf{alphabet}(w)$ denote the set of letters that occur in the string w. By $\mathsf{Prefix}(S)$ and $\mathsf{Suffix}(S)$ we denote the set of prefixes and suffixes resp. of strings from the set S, and $\mathsf{Prefix}_k(S)$ and $\mathsf{Suffix}_k(S)$ denotes their restriction to strings of length k.

2.1 Rewriting and Reachability

A *string rewriting system* over alphabet Σ is a set of rewrite rules. We use standard concepts and notation (see, e. g., Book and Otto [1]) with this extension: A *constrained rule* is a pair of strings l, r, together with a *constraint* $c \in \{\mathsf{factor}, \mathsf{prefix}, \mathsf{suffix}\}$ that indicates where the rule may be applied. The corresponding rewrite relations are

$$\begin{aligned}
&\to_{l,r,\mathsf{factor}} = \{(xly, xry) \mid x, y \in \Sigma^*\},\\
&\to_{l,r,\mathsf{prefix}} = \{(ly, ry) \mid y \in \Sigma^*\},\\
&\to_{l,r,\mathsf{suffix}} = \{(xl, xr) \mid x \in \Sigma^*\}.
\end{aligned}$$

A constrained rule (l, r, c) is denoted by $l \to_c r$. Standard rewriting corresponds to the factor constraint, therefore $\to$ abbreviates $\to_{\mathsf{factor}}$. For a rewrite system R, we define $\to_R$ as the union of the rewrite relations of its rules. For a relation ρ on Σ^* and a set $L \subseteq \Sigma^*$, let $\rho(L) = \{y \mid \exists x \in L, (x, y) \in \rho\}$. Hence the set of *$R$-reachable* strings from L is $\to_R^*(L)$, or $R^*(L)$ for short. A language $L \subseteq \Sigma^*$ is *closed w.r.t. R* if $\to_R(L) \subseteq L$.

▶ **Example 2.1.** For $R = \{cc \to_{\mathsf{factor}} bc, ba \to_{\mathsf{factor}} ac, c \to_{\mathsf{suffix}} bc, b \to_{\mathsf{suffix}} ac\}$, we have $bbb \to_{\mathsf{suffix}} bbac \to_{\mathsf{factor}} bacc$. The reachability set $R^*(\{bc, ac\})$ is $(a + b)b^*c$. This set is closed with respect to R.

A rewriting system R over Σ is called *terminating on* $L \subseteq \Sigma^*$, if for each $w \in L$, each R-derivation starting at w is finite, and R is called *terminating*, written $\mathsf{SN}(R)$, if it is

[1] The Termination Problems Database, Version 10.6, see `http://termination-portal.org/wiki/TPDB`.

terminating on Σ^*. A rewriting system R is called *terminating relative to* a rewriting system S *on* L, if each $(R \cup S)$-derivation starting in L has a only a finite number of R rule applications. If L is not given, we mean Σ^* and write $\mathsf{SN}(R/S)$.

2.2 Forward Closures

Given a rewrite system R over alphabet Σ, a *closure* $C = (l, r)$ of R is a pair of strings with $l \to_R^+ r$ such that each position of r is involved in some step of the derivation. In particular, we use *forward closures* [15].

The set $\mathsf{FC}(R)$ of forward closures of R is defined as the least set of pairs (l, r) of strings that contains R and satisfies

- if $(s, xuy) \in \mathsf{FC}(R)$ and $(u, v) \in \mathsf{FC}(R)$ then $(s, xvy) \in \mathsf{FC}(R)$,
- if $(s, xu) \in \mathsf{FC}(R)$ and $(uy, v) \in \mathsf{FC}(R)$ for $u \neq \epsilon \neq y$ then $(sy, xv) \in \mathsf{FC}(R)$.

The set $\mathsf{FC}(R)$ can also be characterized without recursion in the second partner, as observed by Herrmann [11] in the term rewriting case. This can be used to recursively characterize the set $\mathsf{RFC}(R) = \mathsf{rhs}(\mathsf{FC}(R))$ of right hand sides of forward closures directly [6].

$\mathsf{RFC}(R)$ can also be characterized by factor and suffix rewriting.

▶ **Proposition 2.2.** *$\mathsf{RFC}(R) = (R \cup \mathsf{forw}(R))^*(\mathsf{rhs}(R))$, where*

$$\mathsf{forw}(R) = \{l_1 \to_{\mathsf{suffix}} r \mid (l_1 l_2 \to r) \in R, l_1 \neq \epsilon \neq l_2\}.$$

They are related to termination by

▶ **Theorem 2.3** ([2]). *R is terminating on Σ^* if and only if R is terminating on $\mathsf{RFC}(R)$.*

For a self-contained proof see Section 6 in [26].

▶ **Example 2.4.** For $R = \{cc \to bc, ba \to ac\}$ we have $\mathsf{forw}(R) = \{c \to_{\mathsf{suffix}} bc, b \to_{\mathsf{suffix}} ac\}$ and $\mathsf{RFC}(R) = (a + b)b^*c$, cf. Example 2.1. As $\mathsf{RFC}(R)$ contains no R-redex, R is trivially terminating on $\mathsf{RFC}(R)$, therefore R is terminating by Theorem 2.3.

Later in the paper, we use tiled rewriting to approximate $\mathsf{RFC}(R)$, and we obtain the termination proof of Example 2.4 automatically, see Examples 3.10 and 4.3.

2.3 Partial Algebras and Partial Models

We will recall concepts and notation from [3]. A *partial* Σ*-algebra* $\mathcal{A} = (A, [\![\cdot]\!])$ consists of a non-empty set A and for each n-ary $f \in \Sigma$ a partial function $[\![f]\!] : A^n \rightharpoonup A$. Given $\mathcal{A}$ and a partial assignment of variables $\alpha : V \rightharpoonup A$, the *interpretation* $[\![t, \alpha]\!]$ of $t \in \mathsf{Term}(\Sigma, V)$ is defined as usual, noting that it will not always be defined. If t is ground, we simply write $[\![t]\!]$. A partial algebra is a *partial model* of a rewrite system R if for each rewrite rule $(l \to r) \in R$, and each assignment $\alpha : \mathsf{Var}(l) \to A$, definedness of $[\![l, \alpha]\!]$ implies $[\![l, \alpha]\!] = [\![r, \alpha]\!]$. For a partial Σ-algebra $\mathcal{A} = (A, [\![\cdot]\!])$, a term $t \in \mathsf{Term}(\Sigma, X)$, and a partial assignment $\alpha : \mathsf{Var}(t) \rightharpoonup A$, let $[\![t, \alpha]\!]^*$ denote the set of defined values of subterms of t under α, i. e., $\{[\![s, \alpha]\!] \mid s \trianglelefteq t \wedge [\![s, \alpha]\!] \text{ is defined}\}$. For $T \subseteq A$, let $\mathsf{Lang}_{\mathcal{A}}(T)$ denote the set ot ground terms that can be evaluated inside T, i. e., $\{t \in \mathsf{Term}(\Sigma) \mid [\![t]\!]^* \subseteq T\}$, and let $\mathsf{Lang}_{\mathcal{A}} = \mathsf{Lang}_{\mathcal{A}}(A)$. Note that a partial algebra *is* a deterministic (tree) automaton with set of states A, and partiality means that the automaton may be incomplete.

2.4 Languages defined by Tilings

A strictly locally testable language is specified by considering prefixes, factors, and suffixes of bounded length, called tiles. We give an equivalent definition that allows a uniform description, using end markers $\triangleleft, \triangleright \notin \Sigma$. A similar formalization is employed for two-dimensional tiling in [8].

▶ **Definition 2.5.** *For an alphabet Γ, $k \geq 1$ and $a_i \in \Gamma$, the* k-tiled version *of a string $a_1 \dots a_n$ is the string over Γ^k of all k-tiles, i. e., factors of length k:*

$$\mathit{tiled}_k(a_1 \dots a_n) = [a_1 \dots a_k, a_2 \dots a_{k+1}, \dots, a_{n-k+1} \dots a_n]$$

This string is empty in case $n < k$. Let $\mathit{tiles}_k(w)$ denote $\mathit{alphabet}(\mathit{tiled}_k(w))$.

▶ **Definition 2.6.** *For $k \geq 0$ and $w \in \Sigma^*$, the* k-bordered version *of w is $\mathit{bord}_k(w) = \triangleleft^k w \triangleright^k$ over $\Sigma \cup \{\triangleleft, \triangleright\}$. By $\mathit{btiled}_k(w)$ we abbreviate $\mathit{tiled}_k(\mathit{bord}_{k-1}(w))$, and $\mathit{btiles}_k(w)$ stands for $\mathit{alphabet}(\mathit{btiled}_k(w))$.*

▶ **Example 2.7.** $\mathsf{btiled}_2(abbb) = \mathsf{tiled}_2(\mathsf{bord}_1(abbb)) = \mathsf{tiled}_2(\triangleleft abbb\triangleright) = [\triangleleft a, ab, bb, bb, b\triangleright]$, thus $\mathsf{btiles}_2(abbb) = \{\triangleleft a, ab, bb, b\triangleright\}$. Further, $\mathsf{btiles}_2(\epsilon) = \{\triangleleft\triangleright\}$, $\mathsf{btiles}_2(a) = \{\triangleleft a, a\triangleright\}$, and $\mathsf{btiled}_3(a) = [\triangleleft\triangleleft a, \triangleleft a\triangleright, a\triangleright\triangleright]$.

▶ **Definition 2.8.** *For $k \geq 1$, the language defined by a set of tiles $T \subseteq \mathit{btiles}_k(\Sigma^*)$ is*

$$\mathit{Lang}(T) = \{w \in \Sigma^* \mid \mathit{btiles}_k(w) \subseteq T\}.$$

This is a characterization of the class of strictly locally k-testable languages [16, 23], a subclass of regular languages.

▶ **Example 2.9.** For $k = 2$ and $T = \{\triangleleft a, ab, ba, a\triangleright\}$ we obtain $\mathsf{Lang}(T) = a(ba)^*$.

3 Tiled Rewrite Systems and Shift Algebras

We apply the method of semantic labelling w.r.t. a partial model to transform a local termination problem to a global one. Our contribution is to use the *k-shift algebra*. We obtain Algorithm 4.1 that over-approximates reachability sets w.r.t. rewriting, and is guaranteed to halt.

One application is to approximate right-hand sides of forward closures, to prove global termination (Algorithm 5.1). Later, we approximate right-hand sides of overlap closures to prove relative termination (Algorithm 8.1).

Our intended application area is string rewriting. For proving correctness we want to use concepts and results from local termination [3], so we need a translation to term rewriting. We view strings as terms with unary symbols, and a nullary symbol (representing ϵ), where the rightmost (!) position in the string is the topmost position in the term. As in [3], we choose this order (left to right in the string means bottom to top in the term) since we later use deterministic automata, working from left to right on the string, realising evaluation in the algebra, which goes bottom to top. This choice also has the notational consequence that a string rewriting rule, e. g., $ab \to baa$, is translated to a term rewriting rule $((z)a)b \to ((z)b)a)a$, where z is a variable, which we abbreviate to $(z)ab \to (z)baa$. This is just postfix notation for function application, recommended also by Sakarovitch [18], p. 12.

▶ **Definition 3.1** (The *k-shift algebra*). *For $T \subseteq \mathit{btiles}_k(\Sigma^*)$, the partial algebra $\mathit{Shift}_k(T)$ over signature $\Sigma \cup \{\epsilon, \triangleright\}$ has domain $\mathit{tiles}_{k-1}(\triangleleft^*\Sigma^*\triangleright^*)$, the interpretation of ϵ is $\triangleleft^{k-1}$, and each letter (unary symbol) $c \in \Sigma \cup \{\triangleright\}$ is interpreted by the unary function that maps p to $\mathit{Suffix}_{k-1}(pc)$ if $pc \in T$, and is undefined otherwise.*

We have the following obvious connection (modulo the translation between words and terms) between the language of the algebra (i. e., all terms that have a defined value) and the language of the set of tiles (i. e., all words that can be covered):

▶ **Proposition 3.2.** *For any set of k-tiles T, $\mathsf{Lang}_{\mathsf{Shift}_k(T)} = \mathsf{Prefix}(\mathsf{Lang}(T) \cdot \triangleright^{k-1})$.*

We need the prefix closure since a language of a partial algebra always is subterm-closed, according to the definition from [3], a feature that had already been criticised in [4].

To apply semantic labelling, we need a partial algebra that is a partial model. A k-shift algebra is a model for a rewrite system R only if R does not change the $k-1$ topmost symbols. This property can be guaranteed by the following closure operation that also translates our notion of constrained string rewriting to the standard notion of term rewriting:

▶ **Definition 3.3.** *For a constrained string rewriting system R over Σ define its* context closure*, the term rewriting system $\mathsf{CC}_k(R)$ over $\Sigma \cup \{\epsilon, \triangleright\}$, where ϵ is a constant, all other symbols are unary, and z is a variable symbol, as follows. Note that the second subset consists of ground rules.*

$$\begin{aligned}\mathsf{CC}_k(R) = &\{(z)ly \to (z)ry \mid (l \to_{\mathsf{factor}} r) \in R, y \in \mathsf{tiles}_{k-1}(\Sigma^*\triangleright^*)\} \cup \\ &\{(\epsilon)ly \to (\epsilon)ry \mid (l \to_{\mathsf{prefix}} r) \in R, y \in \mathsf{tiles}_{k-1}(\Sigma^*\triangleright^*)\} \cup \\ &\{(z)ly \to (z)ry \mid (l \to_{\mathsf{suffix}} r) \in R, y = \triangleright^{k-1}\}\end{aligned}$$

Constrained rewrite steps of R on Σ^* are directly related to term rewrite steps of the context closure of R on (the set of terms corresponding to) $\Sigma^*\triangleright^{k-1}$:

▶ **Proposition 3.4.** *$s \to_R t$ iff $(\epsilon)s\triangleright^{k-1} \to_{\mathsf{CC}_k(R)} (\epsilon)t\triangleright^{k-1}$.*

Since $\mathsf{CC}_k(R)$ does keep the $k-1$ topmost (rightmost) symbols intact, the shift algebra of T is a partial model provided it contains a sufficiently large set of tiles:

▶ **Proposition 3.5.** *For a set of k-tiles T and a rewriting system R, if $\mathsf{Lang}_{\mathsf{Shift}_k(T)}$ is closed with respect to R, then $\mathsf{Shift}_k(T)$ is a partial model for $\mathsf{CC}_k(R)$.*

In Section 4 we provide an algorithm for constructing such a closed set T.

Given a partial model, we use it for semantic labeling. The labeling of $\mathsf{CC}_k(R)$ with respect to $\mathsf{Shift}_k(T)$ (see [3], Def. 6.3) produces a term rewriting system that can be re-transformed to a string rewriting system by replacing each function symbol c, that is labelled with an element p from the algebra, to the string (the tile) pc. The following definition avoids the round-trip, and shows how to label the string rewriting system directly.

▶ **Definition 3.6.** *For a rule $l \to_c r$ over signature Σ with $c \in \{\mathsf{factor}, \mathsf{prefix}, \mathsf{suffix}\}$, we define a set of rules over signature $\mathsf{btiles}_k(\Sigma^*)$ by*

$$\begin{aligned}\mathsf{btiled}_k(l \to_{\mathsf{factor}} r) &= \{\mathsf{tiled}_k(xly) \to_{\mathsf{factor}} \mathsf{tiled}_k(xry) \mid x \in T^{\triangleleft}, y \in T^{\triangleright}\} \\ \mathsf{btiled}_k(l \to_{\mathsf{prefix}} r) &= \{\mathsf{tiled}_k(xly) \to_{\mathsf{prefix}} \mathsf{tiled}_k(xry) \mid x = \triangleleft^{k-1}, y \in T^{\triangleright}\} \\ \mathsf{btiled}_k(l \to_{\mathsf{suffix}} r) &= \{\mathsf{tiled}_k(xly) \to_{\mathsf{suffix}} \mathsf{tiled}_k(xry) \mid x \in T^{\triangleleft}, y = \triangleright^{k-1}\}\end{aligned}$$

where $T^{\triangleleft} = \mathsf{tiles}_{k-1}(\triangleleft^*\Sigma^*)$, $T^{\triangleright} = \mathsf{tiles}_{k-1}(\Sigma^*\triangleright^*)$, and for a set of tiles $T \subseteq \mathsf{btiles}_k(\Sigma^*)$ let

$$\mathsf{btiled}_T(l \to_c r) = \mathsf{btiled}_k(l \to_c r) \cap T^* \times T^* \times \{c\},$$

the set of tiled rules that use tiles from T only. Both btiled_k and btiled_T are extended to sets of rules. Note that $\{\triangleleft^{k-1}\} = \mathsf{tiles}_{k-1}(\triangleleft^*)$ and $\{\triangleright^{k-1}\} = \mathsf{tiles}_{k-1}(\triangleright^*)$.

▶ **Example 3.7.** The set $\mathsf{btiled}_2(ba \to_{\mathsf{factor}} ac)$ contains 16 rules, among them $[\lhd b, ba, a\rhd] \to [\lhd a, ac, c\rhd]$, $[\lhd b, ba, aa] \to [\lhd a, ac, ca], \ldots, [ab, ba, a\rhd] \to [aa, ac, c\rhd], \ldots, [cb, ba, ac] \to [ca, ac, cc]$, and $\mathsf{btiled}_2(b \to_{\mathsf{suffix}} ac) = \{[\lhd b, b\rhd] \to [\lhd a, ac, c\rhd], [ab, b\rhd] \to [aa, ac, c\rhd], [bb, b\rhd] \to [ba, ac, c\rhd], [cb, b\rhd] \to [ca, ac, c\rhd]\}$. For $S = \{ac, ba, bb, cc\}$ we get $\mathsf{btiled}_S(ba \to_{\mathsf{factor}} ac) = \{[bb, ba, ac] \to [ba, ac, cc]\}$ and for any strict subset T of S, $\mathsf{btiled}_T(ba \to_{\mathsf{factor}} ac) = \emptyset$.

This translation is faithful, in the following sense:

▶ **Proposition 3.8.** *$\mathsf{btiled}_T(R)$ is exactly the (string rewriting translation of the) labeling of $\mathsf{CC}_k(R)$ with respect to $\mathsf{Shift}_k(T)$.*

To actually enumerate $\mathsf{btiled}_T(R)$ in an implementation, we will fuse both parts of Definition 3.6 by restricting contexts x and y to be elements of T^* right from the beginning.

▶ **Theorem 3.9.** *For $k \geq 1$ and $T \subseteq \mathsf{btiles}_k(\Sigma^*)$, if $\mathsf{Lang}(T)$ is closed with respect to R, then R is terminating on $\mathsf{Lang}(T)$ if and only if $\mathsf{btiled}_T(R)$ is terminating.*

Proof. By Proposition 3.8 and Theorem 6.4 from [3], applicable due to Proposition 3.5. ◀

▶ **Example 3.10** (Example 2.4 continued)**.** Let $R = \{cc \to bc, ba \to ac\}$. Then $\mathsf{RFC}(R) = \mathsf{Lang}(T)$ for the set of tiles $T = \{\lhd a, \lhd b, ab, ac, bb, bc, c\rhd\}$. The set $\mathsf{RFC}(R)$ is closed w.r.t. R by definition and $\mathsf{tiled}_T(R)$ is empty, therefore terminating. By Theorem 3.9, R is terminating on $\mathsf{RFC}(R)$, thus by Theorem 2.3, R is terminating. See Example 4.3 for a computation that produces T from R.

4 Completion in Shift Algebras

To apply Theorem 3.9, we need an R-closed set T of tiles. The following algorithm computes such a set by starting from an initial set S, and successively adding tiles that become reachable via R-steps. This is similar to other algorithms that produce rewrite-closed automata [5]. Due to the algebra we use, we have the stronger property of guaranteed termination.

▶ **Algorithm 4.1.**

- *Specification:*
  - *Input: A term rewriting system R over Σ, a finite partial Σ-algebra $\mathcal{A} = (A, [\![\cdot]\!])$, a set $S \subseteq A$.*
  - *Output: A minimal set $T \subseteq A$ such that $S \subseteq T$ and $\mathsf{Lang}_{\mathcal{A}}(T)$ is closed w.r.t. R.*
- *Implementation: Let $T = \bigcup_i T_i$ for the sequence $S = T_0 \subseteq T_1 \subseteq \cdots$ where*

$$T_{i+1} = T_i \cup \bigcup \{[\![r, \alpha]\!]^* \mid (l \to r) \in R, \alpha : \mathsf{Var}(l) \to T_i, [\![l, \alpha]\!]^* \subseteq T_i\}$$

where it is sufficient to compute a finite prefix.

Proof. This algorithm terminates, since the sequence T_i is increasing, and bounded from above by A, so it is eventually constant. The result is R-closed by construction. ◀

This algorithm will be applied to $\mathsf{CC}_k(R)$, and we construct the context closure on the fly: in each step, we use only those contexts that are accessible in $\mathsf{Shift}_k(T_i)$.

Let us first specify the representation of $\mathsf{Shift}_k(T)$. This algebra is a deterministic automaton, possibly incomplete.

▶ **Definition 4.2.** *For $k \geq 1$, a finite automaton over alphabet $\Sigma \cup \{\lhd, \rhd\}$ is a k-*shift automaton *if its states are in* $\mathsf{tiles}_{k-1}(\lhd^*\Sigma^*\rhd^*)$*, its initial state is* $\lhd^{k-1}$*, its final state is* $\rhd^{k-1}$*, and for each transition* $p \xrightarrow{c} q$*, state q is the suffix of length $k-1$ of pc. Such an automaton A represents the set of tiles (of length k)* $\mathsf{tiles}(A) = \{pc \mid p \xrightarrow{c}_A q\}$.

Condition $[\![l, \alpha]\!]^* \subseteq T_i$ of Algorithm 4.1 is equivalent to the existence of a path in the automaton T_i that starts at state $p = \alpha(z)$ and is labelled l. We call this a *redex path* $p \xrightarrow{l} q$. Adding tiles then corresponds to adding edges and states. Whenever we add edges for some reduct path $p \xrightarrow{r} q'$, corresponding to $[\![r, \alpha]\!]^* \subseteq T_i$, the target state of each transition is determined by the shift property of the automaton. This is in contrast to other completion methods where there is a choice of adding fresh states, or re-using existing states.

The set of states could be defined to be $\mathsf{btiles}_{k-1}(\Sigma^*)$ in advance, but for efficiency, we only store accessible states, and add states as soon as they become accessible.

With the automata representation, we implement $\mathsf{btiled}_T(R)$ as follows: To determine xly in Definition 3.6, we compute all pairs p, q of states with $p \xrightarrow{l} q$. This can be done by starting at each p, but our implementation uses the product-of-relations method of [22]. Note that p, the state where the redex path starts, *is* actually x, the left context.

From state q, we follow all paths of length $k-1$ to determine the set of y (right contexts). For each such pair (x, y), we add the path starting at x labeled ry. Note that this path (for the context-closed reduct) meets the path for ly (the context-closed redex) in the end, since the automaton is a shift automaton. The tree search for possible y can be cut short if we detect that these paths meet earlier.

The following example demonstrates completion only. For examples that use the completed automaton for semantic labeling, see Section 5.

▶ **Example 4.3** (Example 3.10 continued)**.** In order to illustrate the use of shift automata for implementing Algorithm 4.1, consider again $R = \{cc \to bc, ba \to ac\}$ with $\mathsf{forw}(R) = \{c \to_{\mathsf{suffix}} bc, b \to_{\mathsf{suffix}} ac\}$. We choose $k = 2$ and represent $\mathsf{btiled}_2(\mathsf{rhs}(R)) = \{[\lhd b, bc, c\rhd], [\lhd a, ac, c\rhd]\}$ by the left automaton in Figure 1. Here, completion refers to the set of rules $C = \mathsf{CC}_2(R \cup \mathsf{forw}(R)) = \{ccy \to bcy, bay \to acy, c\rhd \to bc\rhd, b\rhd \to ac\rhd \mid y \in \{a, b, c, \rhd\}\}$. In the initial automaton we look for paths of the form $p \xrightarrow{l} q$ for some rule $l \to r \in C$. Two such paths exist, $a \xrightarrow{c\rhd} \rhd$ and $b \xrightarrow{c\rhd} \rhd$. Completion therefore adds the paths $a \xrightarrow{bc\rhd} \rhd$ and $b \xrightarrow{bc\rhd} \rhd$ for the corresponding right-hand sides, resulting in the new edges $a \xrightarrow{b} b$ and $b \xrightarrow{b} b$ (and no new nodes), depicted by the right automaton A. No further completion steps are possible, thus $\mathsf{RFC}(R) \subseteq \mathsf{Lang}(\mathsf{tiles}(A))$ with $\mathsf{tiles}(A) = \{\lhd a, \lhd b, ab, ac, bb, bc, c\rhd\}$. Note that for this simple example, $\subseteq$ could be replaced by equality, but in general the algorithm yields an over-approximation.

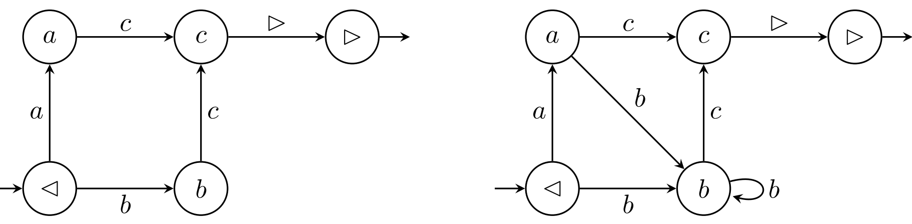


Figure 1 Constructing the shift automaton for $\mathsf{RFC}(\{cc \to bc, ba \to ac\})$ for $k = 2$.

5 Examples of Termination Proofs via Forward Closures

We transform a termination problem as follows:

▶ **Algorithm 5.1.**
- *Specification:*
  - *Input: A rewriting system R over Σ, a number k*
  - *Output: A rewriting system R' over $\mathsf{btiled}_k(\Sigma)$ such that $\mathsf{SN}(R) \iff \mathsf{SN}(R')$*
- *Implementation (and correctness): By Theorem 2.3, $\mathsf{SN}(R)$ iff $\mathsf{SN}(R)$ on $\mathsf{RFC}(R)$. By Proposition 2.2, $\mathsf{RFC}(R) = (R \cup \mathsf{forw}(R))^*(\mathsf{rhs}(R))$. By Algorithm 4.1, we construct T such that $\mathsf{Lang}(T)$ contains $\mathsf{rhs}(R)$ and is closed w.r.t. $R \cup \mathsf{forw}(R)$, that is, $\mathsf{RFC}(R) \subseteq \mathsf{Lang}(T)$. We then use the algebra $\mathsf{Shift}(T)$ as a partial model for $\mathsf{CC}_k(R)$. By Theorem 3.9 and the previous, $\mathsf{SN}(R)$ iff $\mathsf{SN}(\mathsf{btiled}_T(R))$.*

This approach had already been described in [3], Section 8, but there it was left open how to find a suitable partial algebra. An implementation used a finite-domain constraint solver, but then only small domains could be handled. In the present paper, we instead construct a suitable k-shift algebra by completion. Even if it is large, it might help solve the termination problem, cf. Example 5.6 below. We give a few smaller examples first.

▶ **Example 5.2.** We apply Algorithm 5.1 with $k = 3$ to $R = \{ab^3 \to bbaab\}$. We obtain 11 reachable tiles and 12 labeled rules. All of them can be removed by weights. We start with the automaton for $\mathsf{btiled}_3(bbaab)$ (solid edges in Figure 2).

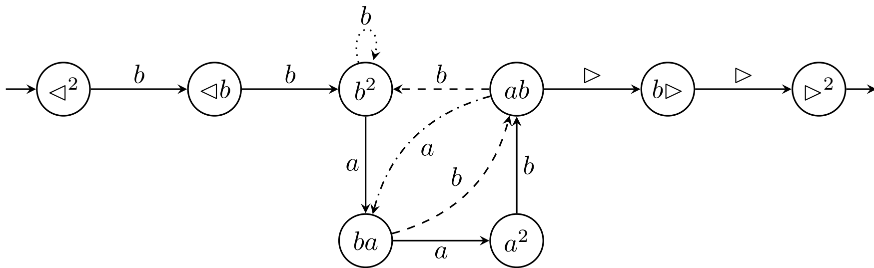


Figure 2 The 3-shift automaton for $\mathsf{RFC}(ab^3 \to bbaab)$.

It contains no R-redex. There is a $\mathsf{forw}(R)$-redex for $ab \to_{\mathsf{suffix}} bbaab$ starting at ba. We add a reduct path, starting with two fresh (dashed) edges. This creates a $\mathsf{forw}(R)$-redex for $ab \to_{\mathsf{suffix}} bbaab$ from b^2. To cover this, we add the loop at b^2 (dotted). Now we have a R-redex $ba \to a^2 \to ab \to b^2 \to b^2$. The corresponding reduct path is $ba \to ab \to b^2 \to ba \to a^2 \to ab$. The redex needs to be right-context-closed with a, and with b, as these are the possible continuations from b^2. So we context-close the reduct path as well, adding one more edge $ab \xrightarrow{a} ba$ (dash-dotted), as $ab \xrightarrow{b} b^2$ is already present. This introduces an R-redex from ab to b^2, with right extensions a and b. The extended reduct paths are already present. The automaton is now closed with respect to $R \cup \mathsf{forw}(R)$. It represents the set of tiles

$$T = \{\triangleleft\triangleleft b, \triangleleft bb, bba, bbb, baa, bab, aab, aba, abb, ab\triangleright, b\triangleright\triangleright\}.$$

Absent from T are
- $\triangleleft\triangleleft\triangleright, \triangleleft\triangleright\triangleright, \triangleleft\Sigma\triangleright$ (meaning that $\mathsf{RFC}(R)$ does not contain strings of length 0 or 1),
- as well as $\triangleleft a\Sigma, \triangleleft ba, \Sigma a\triangleright$ (meaning that $\mathsf{RFC}(R)$ starts with b^2 and ends with b),
- and a^3 (meaning that $\mathsf{RFC}(R)$ does not have a^3 as a factor).

Finally, we compute $\mathsf{btiled}_T(R)$. There are three R-redex paths in the automaton, starting at b^2, ba, ab, respectively, and all ending in b^2. They will be right-context-closed by Σ^2, resulting in the following $3 \times 2^2 = 12$ tiled rules, where $x, y \in \Sigma$:

$$[bba, bab, abb, b^3, bbx, bxy] \to [b^3, b^3, bba, baa, aab, abx, bxy]$$
$$[baa, aab, abb, b^3, bbx, bxy] \to [bab, abb, bba, baa, aab, abx, bxy]$$
$$[aba, bab, abb, b^3, bbx, bxy] \to [abb, b^3, bba, baa, aab, abx, bxy]$$

With the following weights, all rules are strictly decreasing:

$$bbb \mapsto 8, bab \mapsto 4, abb \mapsto 3, bba \mapsto 3, \text{others} \mapsto 0.$$

This shows termination of $\mathsf{btiled}_T(R)$, thus, of R.

The following observation, similar to *semantic unlabeling* [20], allows to use the partial algebra for removing rules without labeling:

▶ **Proposition 5.3.** *If the set of tiles T is R-closed, and $R_0 \subseteq R$ such that $\mathsf{tiled}_T(R_0) = \emptyset$, then $\mathsf{SN}(R)$ on $\mathsf{Lang}(T)$ if and only if $\mathsf{SN}(R \setminus R_0)$ on $\mathsf{Lang}(T)$.*

Proof. Let $R_1 = R \setminus R_0$. By Theorem 3.9, each R-derivation corresponds to a $\mathsf{btiled}_T(R)$-derivation. By assumption, this is a $\mathsf{btiled}_T(R_1)$-derivation. This can be mapped back to a R_1-derivation, using the same theorem. ◀

If R_0 is nonempty, this produces a strictly smaller termination problem on the original alphabet. This results in a modification of Algorithm 5.1:

▶ **Algorithm 5.4.**

- *Specification:*
  - *Input: A rewriting system R over Σ, a number k*
  - *Output: A rewriting system R' over Σ such that $\mathsf{SN}(R) \iff \mathsf{SN}(R')$, or failure.*
- *Implementation: By Algorithm 4.1, construct T such that $\mathsf{Lang}(T)$ contains $\mathsf{rhs}(R)$ and is closed w.r.t. $R \cup \mathsf{forw}(R)$, that is, $\mathsf{RFC}(R) \subseteq \mathsf{Lang}(T)$. Let $R_0 \subseteq R$ consist of all rules $(l \to r) \in R$ with $\mathsf{btiled}_T(l \to r) = \emptyset$. If $\emptyset \neq R_0$, then output $R \setminus R_0$, else fail.*

▶ **Example 5.5.** We apply Algorithm 5.4, for $k = 2$, to $R = \{ab \to bca, bc \to cbb, ba \to acb\}$. This is `SRS/Zantema/z018` from TPDB. We construct the 2-shift automaton, see Figure 3, and we find that $\mathsf{btiled}_T(ab \to bca) = \emptyset$. The algorithm outputs $\{bc \to cbb, ba \to acb\}$. Note that

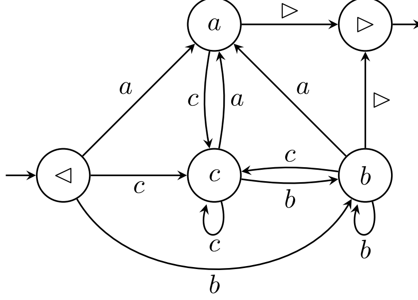


Figure 3 The 2-shift automaton for $\mathsf{RFC}(z018)$.

the automaton contains redexes for $(a \to_{\mathsf{suffix}} bca) \in \mathsf{forw}(ab \to bca)$ (from states $\lhd, c$, and b) but the criterion is the occurrence of $ab \to bca$ only. To handle the resulting termination problem, we reverse all strings in all (remaining) rules, obtaining $\{cb \to bbc, ab \to bca\}$. Again we apply Algorithm 5.4 and this time we find that ab does not occur in the automaton. This leaves $\{cb \to bbc\}$. Applying the algorithm one more time, we find that there is no cb in the 2-shift automaton for $\mathsf{RFC}(cb \to bbc)$. The algorithm outputs $\emptyset$, and we have proved termination of z018.

▶ **Example 5.6.** We prove termination of Zantema's problem $\{a^2b^2 \to b^3a^3\}$, a classical benchmark. We give an outline of the proof that consists of a chain of transformations. Each node (r, s) denotes a rewrite system with r rules on s letters. The arrows $\xrightarrow{\mathsf{RFC}_k}_{\mathsf{All}}$ and $\xrightarrow{\mathsf{RFC}_k}_{\mathsf{Rem}}$ denote application of Algorithm 5.1 and Algorithm 5.4 respectively, and $\overset{\mathsf{W}}{\to}$ denotes removal of rules by weights.

$$(1,2) \xrightarrow[\mathsf{All}]{\mathsf{RFC}_2} (4,4) \xrightarrow[\mathsf{Rem}]{\mathsf{RFC}_5} (3,4) \xrightarrow[\mathsf{All}]{\mathsf{RFC}_2} (12,8) \xrightarrow[\mathsf{All}]{\mathsf{RFC}_3} (105,26) \overset{\mathsf{W}}{\to} (60,26)$$
$$\xrightarrow[\mathsf{Rem}]{\mathsf{RFC}_5} (37,26) \xrightarrow[\mathsf{All}]{\mathsf{RFC}_2} (97,44) \overset{\mathsf{W}}{\to} (65,43) \xrightarrow[\mathsf{Rem}]{\mathsf{RFC}_5} (36,43) \overset{\mathsf{W}}{\to} (28,43) \xrightarrow[\mathsf{All}]{\mathsf{RFC}_2} (86,68)$$
$$\overset{\mathsf{W}}{\to} (50,62) \xrightarrow[\mathsf{All}]{\mathsf{RFC}_3} (246,128) \overset{\mathsf{W}}{\to} (42,84) \xrightarrow[\mathsf{Rem}]{\mathsf{RFC}_5} (2,44) \overset{\mathsf{W}}{\to} (0,0)$$

It is even possible to give a termination proof *without* using weights at all:

$$(1,2) \xrightarrow[\mathsf{All}]{\mathsf{RFC}_2} (4,4) \xrightarrow[\mathsf{Rem}]{\mathsf{RFC}_5} (3,4) \xrightarrow[\mathsf{All}]{\mathsf{RFC}_3} (40,15) \xrightarrow[\mathsf{All}]{\mathsf{RFC}_2} (105,26) \xrightarrow[\mathsf{Rem}]{\mathsf{RFC}_5} (65,26) \xrightarrow[\mathsf{Rem}]{\mathsf{RFC}_5} (52,26)$$
$$\xrightarrow[\mathsf{Rem}]{\mathsf{RFC}_5} (37,26) \xrightarrow[\mathsf{All}]{\mathsf{RFC}_2} (97,44) \xrightarrow[\mathsf{Rem}]{\mathsf{RFC}_5} (37,43) \xrightarrow[\mathsf{Rem}]{\mathsf{RFC}_5} (36,43) \xrightarrow[\mathsf{All}]{\mathsf{RFC}_2} (110,68) \xrightarrow[\mathsf{Rem}]{\mathsf{RFC}_5} (80,64)$$
$$\xrightarrow[\mathsf{All}]{\mathsf{RFC}_2} (192,93) \xrightarrow[\mathsf{Rem}]{\mathsf{RFC}_5} (96,89) \xrightarrow[\mathsf{Rem}]{\mathsf{RFC}_3} (58,79) \xrightarrow[\mathsf{Rem}]{\mathsf{RFC}_5} (32,66) \xrightarrow[\mathsf{Rem}]{\mathsf{RFC}_3} (0,0).$$

▶ **Example 5.7.** We show that our method can be applied as a preprocessor for other termination provers. We consider $R = \{0000 \to 1001, 0101 \to 0010\}$, which is `SRS/Gebhardt/16` from the TPDB. After the chain of transformations

$$(2,2) \xrightarrow[\mathsf{All}]{\mathsf{RFC}_3} (98,20) \overset{\mathsf{W}}{\to} (24,11) \xrightarrow[\mathsf{Rem}]{\mathsf{RFC}_2} (17,10) \overset{\mathsf{W}}{\to} (15,8),$$

the resulting problem can be solved by $\mathsf{T_TT_2}$ [14] quickly, via KBO. $\mathsf{T_TT_2}$ did not solve this problem in the Termination Competition 2018.

6 Overlap Closures and Relative Termination

We now apply our approach to prove relative termination. With relative termination, the RFC method does not work.

▶ **Example 6.1.** R/S may nonterminate although R/S terminates on $\mathsf{RFC}(R \cup S)$. For example, let $R = \{ab \to a\}$ and $S = \{c \to bc\}$. We have $\mathsf{RFC}(R \cup S) = a \cup b^+c$. This does not have a factor ab, therefore $\mathsf{SN}(R/S)$ on $\mathsf{RFC}(R \cup S)$. On the other hand, $\neg\mathsf{SN}(R/S)$ because of the loop $\underline{ab}c \to_R a\underline{c} \to_S abc$.

Therefore, we use overlap closures instead. To prove correctness of this approach, we use a characterization of overlap closures as derivations in which every position between letters is touched. A new left-recursive characterization of overlap closures (Corollary 7.1) allows us to enumerate $\mathsf{ROC}(R)$ by completion.

Let $\mathsf{OC}(R)$ denote the set of overlap closures [10], and let $\mathsf{ROC}(R) = \mathsf{rhs}(\mathsf{OC}(R))$. A position between letters in the starting string of a derivation is called *touched* by the derivation if it has no residual in the final string.

▶ **Example 6.2.** For the rewrite system $R = \{ab \to baa\}$ over alphabet $\{a, b\}$, all positions labelled by | in the starting string $a|a|ba|b$ are touched by the derivation $aabab \to_R abaaab \to_R baaaaab \to_R baaaabaa$. The position between b and a in the starting string has the residual position between a and b in the final string.

▶ **Lemma 6.3.** *[7, Lemma 3] The set $OC(R)$ of overlap closures of R is the set of all R-derivations where all initial positions between letters are touched.*

Termination has been characterized by forward closures ([2]). In the following we obtain a characterization of relative termination by overlap closures.

▶ **Definition 6.4.** *For a finite or infinite R-derivation A, let $\mathit{Inf}(A)$ denote the set of rules that are applied infinitely often in A. (For a finite derivation, $\mathit{Inf}(A) = \emptyset$.)*

▶ **Proposition 6.5.** *For each R-derivation A, there are finitely many R-derivations $B_1, \ldots, B_k$ that start in $ROC(R)$, and $\mathit{Inf}(A) = \bigcup_i \mathit{Inf}(B_i)$.*

Proof. If A is empty, then $k = 0$. If A has a finite prefix that is an OC, then $k = 1$ and B_1 is the (infinite) suffix. Else, the start of A has a position that is never touched during A. We can then split the derivation, and use induction by the length of the start of the derivation. ◀

▶ **Proposition 6.6.** *$SN(R/S)$ if and only if for each $(R \cup S)$-derivation A, $\mathit{Inf}(A) \cap R = \emptyset$.*

The following theorem says that for analysis of relative termination, we can restrict to derivations starting from right-hand sides of overlap closures.

▶ **Theorem 6.7.** *$SN(R/S)$ if and only if $SN(R/S)$ on $ROC(R \cup S)$).*

Proof. The implication from left to right is trivial, as we consider a subset of derivations. For the other direction, let A be an $(R \cup S)$-derivation. Using Proposition 6.5 we obtain $B_1, \ldots, B_k$ for A such that

$$\mathsf{Inf}(A) \cap R = (\bigcup_i \mathsf{Inf}(B_i)) \cap R = \bigcup_i (\mathsf{Inf}(B_i) \cap R) = \bigcup_i \emptyset = \emptyset,$$

thus $\mathsf{SN}(R/S)$ by Proposition 6.6. ◀

7 Computation of Overlap Closures by Completion

We employ the following left-recursive characterisation of $\mathsf{ROC}(R)$ (proved in the Appendix) that is suitable for a completion algorithm.

▶ **Corollary 7.1.** *$ROC(R)$ is the least set S such that*

1. *$rhs(R) \subseteq S$,*
2. *if $tx \in S$ and $(xu, v) \in R$ for some $t, x, u \neq \epsilon$ then $tv \in S$;*
3. *if $xt \in S$ and $(ux, v) \in R$ for some $t, x, u \neq \epsilon$ then $vt \in S$;*
4. *if $tut' \in S$ and $(u, v) \in R$ then $tvt' \in S$;*
5. *if $tx \in S$ and $yv \in S$ and $(xwy, z) \in R$ for some $t, x, y, v \neq \epsilon$ then $tzv \in S$.*

Note that Item 4 is the standard (factor) rewriting relation of R, Item 2 is suffix rewriting with respect to $\mathsf{forw}(R)$, and Item 3 is prefix rewriting with respect to

$$\mathsf{backw}(R) = \{l_2 \to_{\mathsf{prefix}} r \mid (l_1 l_2 \to r) \in R, l_1 \neq \epsilon \neq l_2\}.$$

Item 5 is an inference rule with two premises, and cannot be written as a rewrite relation. As we still want to apply the partial algebra approach, we have two options: modify that approach to allow more premises, or modify our translation, as follows.

Starting from the automaton constructed in Section 3, we add a path from final state $\rhd^{k-1}$ to initial state $\lhd^{k-1}$, consisting of $k-1$ transitions labelled $\lhd$. The language of this automaton is $\mathsf{Lang}(T)\rhd^{k-1}(\lhd^{k-1}\,\mathsf{Lang}(T)\rhd^{k-1})^*$. Note that this is still a shift automaton. Then an application of Item 5 of Corollary 7.1 with $(xwy, t) \in R$ is realized by a standard rewrite step $x\rhd^{k-1}\lhd^{k-1}y \to_{\mathsf{factor}} t$.

Similar to Definition 3.1, Proposition 3.2, Definition 3.3, we have

▶ **Definition 7.2.** *For $T \subseteq \mathsf{btiles}_k(\Sigma^*)$ the* looped *partial algebra $\mathsf{Shift}_k^o(T)$ has signature $\Sigma \cup \{\lhd, \rhd\}$, domain $\mathsf{tiles}_{k-1}((\lhd^{k-1}\Sigma^*\rhd^{k-1})^*)$, the interpretation of ϵ is $\lhd^{k-1}$, and each letter $c \in \Sigma \cup \{\lhd, \rhd\}$ maps p to $\mathsf{Suffix}_{k-1}(pc)$, if $pc \in T \cup \mathsf{tiles}_k(\rhd^{k-1}\lhd^{k-1})$, and is undefined otherwise.*

▶ **Proposition 7.3.** *For a set of tiles T, we have*

$$\mathsf{Lang}_{\mathsf{Shift}_k^o(T)} = \mathsf{Prefix}(\mathsf{Lang}(T)\rhd^{k-1}(\lhd^{k-1}\,\mathsf{Lang}(T)\rhd^{k-1})^*).$$

▶ **Definition 7.4.** *Let $\mathsf{CC}_k^o(R) = \{(z)x\rhd^{k-1}\lhd^{k-1}ye \to (z)re \mid (xwy \to_{\mathsf{factor}} r) \in R, x \neq \epsilon \neq y, e \in \mathsf{tiles}_{k-1}(\Sigma^*\rhd^*)\}$.*

The purpose of this construction is:

▶ **Proposition 7.5.** *For a set of k-tiles T and a rewriting system R, if $\mathsf{Lang}_{\mathsf{Shift}_k^o(T)}$ is closed with respect to $\mathsf{CC}_k(R \cup \mathsf{forw}(R) \cup \mathsf{backw}(R)) \cup \mathsf{CC}_k^o(R)$, then $\mathsf{ROC}(R)\rhd^{k-1} \subseteq \mathsf{Lang}_{\mathsf{Shift}_k^o(T)}$.*

▶ **Example 7.6.** We illustrate the completion algorithm to obtain an approximation for $\mathsf{ROC}(R)$, for $R = \{a^3 \to a^2b^2a^2\}$. We take $k = 4$ and start with the automaton for $\mathsf{rhs}(R)$, and include the backwards path from $\rhd^3$ to $\lhd^3$ (the solid arrows in Figure 4).

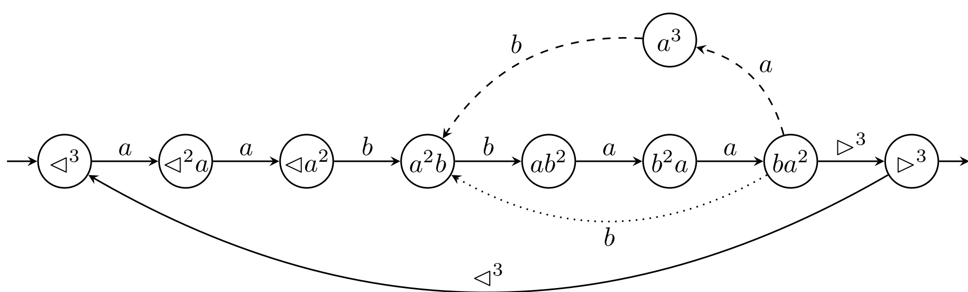


Figure 4 The 4-shift automaton for $\mathsf{ROC}(a^3 \to a^2b^2a^2)$.

We now consider rules $(a\rhd^3\lhd^3ae \to a^2b^2a^2e) \in \mathsf{CC}^o(R)$. These can only start at state b^2a, and the only choice for the right 3-context e in those rules is abb. The reduct path needs two fresh edges (dashed). For rules $(a^2\rhd^3\lhd^3ae \to a^2b^2a^2e) \in \mathsf{CC}^o(R)$, a redex must start in ab^2, and the only right 3-context e is still abb. The reduct path needs one extra edge (dotted). The automaton is now closed also with respect to the other operations. We

compute $\mathsf{btiled}_T(R)$. There is just one R-redex, starting at ab^2, with just one right extension bba. This creates just one labeled rule

$$[abba, bbaa, baaa, aaab, aabb, abba] \to [abba, bbaa, baab, aabb, abba, bbaa, baab, aabb, abba].$$

Of course, the actual implementation will not explicitly represent the path labeled $\lhd^3$. Similar to Theorem 3.9 we have

▶ **Theorem 7.7.** *If $\mathsf{Lang}(T)$ is closed w.r.t. $R \cup S$, then $\mathsf{SN}(R/S)$ on $\mathsf{Lang}(T)$ if and only if $\mathsf{SN}(\mathsf{btiled}_T(R)/\,\mathsf{btiled}_T(S))$.*

For the proof, we need an obvious extension of [3] Thm 6.4 for relative termination, by keeping track of the origin (R or S) of labeled rules.

8 Examples of Relative Termination Proofs via Overlap Closures

We transform global relative termination $\mathsf{SN}(R/S)$ as follows:

▶ **Algorithm 8.1.**

- *Specification:*
  - *Input: rewriting systems R, S over Σ, number k*
  - *Output: rewriting systems R', S' over $\mathsf{btiled}_k(\Sigma)$ such that $\mathsf{SN}(R/S) \iff \mathsf{SN}(R'/S')$.*
- *Implementation (and correctness): By Theorem 6.7, $\mathsf{SN}(R/S)$ iff $\mathsf{SN}(R/S)$ on $\mathsf{ROC}(R\cup S)$. By Corollary 7.1, $\mathsf{ROC}(R)$ is obtained by completion. By Algorithm 4.1, we construct T such that $\mathsf{Lang}(T)$ contains $\mathsf{rhs}(R)$ and is closed w.r.t. $\mathsf{CC}(R \cup S) \cup (\mathsf{forw}(R \cup S) \cup \mathsf{backw}(R \cup S)) \cup \mathsf{CC}^o(R \cup S)$, that is, $\mathsf{ROC}(R) \subseteq \mathsf{Lang}(T)$. By Theorem 3.9 and the previous, $\mathsf{SN}(R/S)$ iff $\mathsf{SN}(\mathsf{btiled}_T(R)/\,\mathsf{btiled}_T(S))$.*

It is often the case that $\mathsf{SN}(\mathsf{btiled}_T(R)/\,\mathsf{btiled}_T(S))$ can be obtained with some easy method, e. g., weights.

Similar to Algorithm 5.4, there is a variant that removes rules in case $\mathsf{btiled}_T(R_0 \cup S_0) = \emptyset$.

▶ **Example 8.2.** $\mathsf{SN}(ababa \to \epsilon/ab \to bbaa)$ (`SRS_Relative/Waldmann_06_relative/r4` from TPDB) can be solved quickly by Algorithm 8.1 with $k = 4$. The tiled system has 270 rules on 33 tiles, and can be solved with weights. Alternatively, tiling of width 5 produces 51 reachable tiles, where the left-hand side of the strict rule is not covered, so can be removed.

In the Termination Competition 2018, AProVE [9] solved this benchmark with double root labeling, which is very similar to tiling of width 3, but this took more than 4 minutes.

▶ **Example 8.3.** The *bowls and beans* problem had been suggested by Vincent van Oostrom [21]. It asks to prove termination of this relation:

If a bowl contains two or more beans, pick any two beans in it and move one of them to the bowl on its left and the other to the bowl on its right.

In a direct model, a configuration is a function $\mathbb{Z} \to \mathbb{N}$ with finite support. In a rewriting model, this is encoded as a string. Several such models have been submitted to TPDB by Hans Zantema (`SRS_Standard/Zantema_06/beans[1..7]`). We consider here a formalisation as a relative termination problem (`SRS_Relative/Waldmann_06_relative/rbeans`).

$$\{baa \to abc, ca \to ac, cb \to ba\}/\{\epsilon \to b\}$$

Here, a is a bean, b separates adjacent bowls, and c transports a bean to the next bowl. The relative rule is used to add extra bowls at either end – although it can be applied anywhere, meaning that any bowl can be split in two, anytime, which does not hurt termination. To the best of our knowledge, this benchmark problem had never been solved in a termination competition. We can now give a termination proof via tiling of width 3, and using overlap closures. This results in a relative termination problem with 560 rules on 47 letters where 305 rules can be removed by weights, and the remaining strict rules by KBO.

The following example applies Algorithm 8.1 to a relative termination problem that comes from the dependency pairs transformation.

▶ **Example 8.4.** The system $\{ababaababa \to abaabababababaab\}$ is part of the enumeration `SRS_Standard/Wenzel_16`, and it was not solved in Termination Competitions up to 2018. In the competition of 2019, Matchbox obtained a termination proof with outline

$$(1,2) \overset{\mathsf{DP}}{\to} (9,3) \xrightarrow[\mathsf{All}]{\mathsf{ROC}_3} (56,17) \overset{\mathsf{W}}{\to} (34,14) \overset{\mathsf{EDG}}{\to} (24,14) \xrightarrow[\mathsf{Rem}]{\mathsf{ROC}_3} (18,10) \xrightarrow[\mathsf{All}]{\mathsf{ROC}_3} (276,46)$$
$$\overset{\mathsf{W}}{\to} (212,39) \overset{\mathsf{EDG}}{\to} (206,39) \xrightarrow[\mathsf{Rem}]{\mathsf{ROC}_3} (151,29) \xrightarrow[\mathsf{All}]{\mathsf{ROC}_3} (2558,138) \overset{\mathsf{W}}{\to} (1962,115)$$
$$\overset{\mathsf{EDG}}{\to} (1960,115) \xrightarrow[\mathsf{Rem}]{\mathsf{ROC}_3} (1082,86) \overset{\mathsf{W}}{\to} (156,44),$$

where $\longrightarrow^{\mathsf{ROC}_k}_{\mathsf{All}}$ and $\longrightarrow^{\mathsf{ROC}_k}_{\mathsf{Rem}}$ denote an application of Algorithm 8.1, or its variant for rule removal, respectively. $\overset{\mathsf{DP}}{\to}$ stands for the dependency pairs transformation, and $\overset{\mathsf{EDG}}{\to}$ denotes the restriction to a strongly connected component of the (estimated) dependency graph. The proof ends successfully with an empty graph.

There are two more systems $\{ababaababa \to abaababababaab\}$ and $\{abaababaab \to aababaabaabab\}$ with the same status. Intermediate systems have up to 3940 rules.

9 Experimental Evaluation

Sparse tiling is implemented in the termination prover Matchbox[2] that won the categories *SRS Standard* and *SRS Relative* in the Termination Competition 2019. Matchbox employs a parallel proof search with a portfolio of algorithms, including Algorithm 8.1.

For relative termination, we use weights, matrix interpretations over the naturals, and tiling of widths 2, 3, 5, 8 (in parallel), cf. Example 8.3. For standard termination, we use RFC matchbounds, and (in parallel) the dependency pairs (DP) transformation, creating a relative termination problem, to which we apply weights, matrix interpretations over natural and arctic numbers, and tiling of width 3 (only), cf. Example 8.4.

Table 1 shows performance of variants of these strategies on SRS benchmarks of TPDB, as measured on Starexec, under the Termination profile (5 minutes wall clock, 20 minutes CPU clock, 128 GByte memory). In all experiments, we keep using weights and (for standard termination) the DP transform. The bottom right entry of each sub-table contains the result for the full strategy, used in competition.

We note a strong increase in the last column (matrices:yes) of the left sub-table. We conclude that sparse tiling is important for relative termination proofs. The right sub-table shows a very weak increase in the corresponding column. We conclude that with Matchbox' current search strategy for standard termination, other methods overshadow tiling, e. g., RFC matchbounds are used in 578 proofs, and arctic matrices in 389 proofs.

[2] `https://gitlab.imn.htwk-leipzig.de/waldmann/pure-matchbox`

Table 1 Number of termination proofs obtained by variants of Matchbox.

SRS Relative Starexec Job 33975		matrices no	matrices yes
tilling	no	1	72
tilling	yes	176	225

SRS Standard Starexec Job 33976		RFC matchbounds, matrices none	RFC matchbounds, matrices both
tilling	no	100	1122
tilling	yes	512	1133

For relative termination, the method of tiling, with weights, but without matrices, is already quite powerful with 176 proofs, a number between those for AProVE (163) and MultumNonMulta (192).

Table 2 shows the widths used in tiling proofs for relative SRS. The sum of the bottom row is larger than the total number of proofs (225) since one proof may use several widths.

Table 2 Number of termination proofs for relative SRS, using given width of tiling.

width	2	3	5	8
proofs	150	57	38	11

We observe that short tiles appear more often. We think the reason is that larger tiles tend to create larger systems that are more costly to handle, while resources (time and space on Starexec) are fixed. This is also the reason for using width 3 only, for standard termination.

10 Conclusion

We have presented *sparse tiling*, a method to compute a regular over-approximation of reachability sets, using sets of tiles, represented as automata, and we applied this to the analysis of termination and relative termination. The method is an instance of semantic labeling via a partial algebra. Our contribution is the choice of the k-shift algebra.

We also provide a powerful implementation in Matchbox that contributed to winning the SRS categories in the Termination Competition 2019. An exact measurement of that contribution is difficult since termination proof search (in Matchbox) depends on too many parameters.

Interesting open questions (that are independent of any implementation) are about the relation between sparse tilings of different widths, and between sparse tilings and other methods, e.g., matchbounds.

Since our focus for the present paper is string rewriting, we also leave open the question of whether sparse tiling would be useful for termination of term rewriting.

References

1 Ronald V. Book and Friedrich Otto. *String-rewriting systems.* Texts and Monographs in Computer Science. Springer, New York, 1993.

2 Nachum Dershowitz. Termination of Linear Rewriting Systems. In Shimon Even and Oded Kariv, editors, *Automata, Languages and Programming, 8th Colloquium, Acre (Akko), Israel, July 13-17, 1981, Proceedings*, volume 115 of *LNCS*, pages 448–458. Springer, 1981. doi:10.1007/3-540-10843-2_36.

3 Jörg Endrullis, Roel C. de Vrijer, and Johannes Waldmann. Local Termination: theory and practice. *Logical Methods in Computer Science*, 6(3), 2010. arXiv:1006.4955.

4. Bertram Felgenhauer and René Thiemann. Reachability, confluence, and termination analysis with state-compatible automata. *Inf. Comput.*, 253:467–483, 2017. doi:10.1016/j.ic.2016.06.011.
5. Thomas Genet. Decidable Approximations of Sets of Descendants and Sets of Normal Forms. In Tobias Nipkow, editor, *Rewriting Techniques and Applications, 9th International Conference, RTA-98, Tsukuba, Japan, March 30 - April 1, 1998, Proceedings*, volume 1379 of *LNCS*, pages 151–165. Springer, 1998. doi:10.1007/BFb0052368.
6. Alfons Geser, Dieter Hofbauer, and Johannes Waldmann. Match-Bounded String Rewriting Systems. *Appl. Algebra Eng. Commun. Comput.*, 15(3-4):149–171, 2004. doi:10.1007/s00200-004-0162-8.
7. Alfons Geser and Hans Zantema. Non-looping string rewriting. *ITA*, 33(3):279–302, 1999. doi:10.1051/ita:1999118.
8. Dora Giammaresi and Antonio Restivo. Two-Dimensional Languages. In Arto Salomaa and Grzegorz Rozenberg, editors, *Handbook of Formal Languages*, volume 3, pages 215–267. Springer, 1997.
9. Jürgen Giesl, Cornelius Aschermann, Marc Brockschmidt, Fabian Emmes, Florian Frohn, Carsten Fuhs, Jera Hensel, Carsten Otto, Martin Plücker, Peter Schneider-Kamp, Thomas Ströder, Stephanie Swiderski, and René Thiemann. Analyzing Program Termination and Complexity Automatically with AProVE. *J. Autom. Reasoning*, 58(1):3–31, 2017. doi:10.1007/s10817-016-9388-y.
10. John V. Guttag, Deepak Kapur, and David R. Musser. On Proving Uniform Termination and Restricted Termination of Rewriting Systems. *SIAM J. Comput.*, 12(1):189–214, 1983. doi:10.1137/0212012.
11. Miki Hermann. *Divergence des systèmes de réécriture et schématisation des ensembles infinis de termes*. Habilitation, Université de Nancy, France, March 1994.
12. Dieter Hofbauer. System description: MultumNonMulta. In A. Middeldorp and R. Thiemann, editors, *15th Intl. Workshop on Termination, WST 2016, Obergurgl, Austria, 2016, Proceedings*, page 90, 2016.
13. Dieter Hofbauer. MultumNonMulta at TermComp 2018. In S. Lucas, editor, *16th Intl. Workshop on Termination, WST 2016, Oxford, U. K., 2018, Proceedings*, page 80, 2018.
14. Martin Korp, Christian Sternagel, Harald Zankl, and Aart Middeldorp. Tyrolean Termination Tool 2. In Ralf Treinen, editor, *Rewriting Techniques and Applications, 20th International Conference, RTA 2009, Brasília, Brazil, June 29 - July 1, 2009, Proceedings*, volume 5595 of *LNCS*, pages 295–304. Springer, 2009. doi:10.1007/978-3-642-02348-4_21.
15. Dallas S. Lankford and D. R. Musser. A finite termination criterion. Technical report, Information Sciences Institute, Univ. of Southern California, Marina-del-Rey, CA, 1978.
16. Robert McNaughton and Seymour Papert. *Counter-Free Automata*. MIT Press, 1971.
17. Aart Middeldorp, Hitoshi Ohsaki, and Hans Zantema. Transforming Termination by Self-Labelling. In Michael A. McRobbie and John K. Slaney, editors, *Automated Deduction - CADE-13, 13th International Conference on Automated Deduction, New Brunswick, NJ, USA, July 30 - August 3, 1996, Proceedings*, volume 1104 of *LNCS*, pages 373–387. Springer, 1996. doi:10.1007/3-540-61511-3_101.
18. Jacques Sakarovitch. *Éléments the théorie des automates*. Vuibert Informatique, 2003.
19. Christian Sternagel and Aart Middeldorp. Root-Labeling. In Andrei Voronkov, editor, *Rewriting Techniques and Applications, 19th International Conference, RTA 2008, Hagenberg, Austria, July 15-17, 2008, Proceedings*, volume 5117 of *LNCS*, pages 336–350. Springer, 2008. doi:10.1007/978-3-540-70590-1_23.
20. Christian Sternagel and René Thiemann. Modular and Certified Semantic Labeling and Unlabeling. In Manfred Schmidt-Schauß, editor, *Proceedings of the 22nd International Conference on Rewriting Techniques and Applications, RTA 2011, May 30 - June 1, 2011, Novi Sad, Serbia*, volume 10 of *LIPIcs*, pages 329–344. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011. doi:10.4230/LIPIcs.RTA.2011.329.

21 Vincent van Oostrom. Bowls and Beans. CWI puzzle, http://www.phil.uu.nl/~oostrom/publication/misc.html, accessible via https://web.archive.org/, 2004.

22 Johannes Waldmann. Efficient Completion of Weighted Automata. In Andrea Corradini and Hans Zantema, editors, *Proceedings 9th International Workshop on Computing with Terms and Graphs, TERMGRAPH 2016, Eindhoven, The Netherlands, April 8, 2016.*, volume 225 of *EPTCS*, pages 55–62, 2016. doi:10.4204/EPTCS.225.8.

23 Yechezkel Zalcstein. Locally testable languages. *Journal of Computer and System Sciences*, 6(2):151–167, 1972. doi:10.1016/S0022-0000(72)80020-5.

24 Hans Zantema. Termination of Term Rewriting by Semantic Labelling. *Fundam. Inform.*, 24(1/2):89–105, 1995. doi:10.3233/FI-1995-24124.

25 Hans Zantema. Termination. In Terese, editor, *Term Rewriting Systems*, pages 181–259. Cambrigde Univ. Press, 2003.

26 Hans Zantema. Termination of String Rewriting Proved Automatically. *J. Autom. Reasoning*, 34(2):105–139, 2005. doi:10.1007/s10817-005-6545-0.

A Composition Trees of Overlap Closures

In this section we derive a left-recursive characterization of overlap closures in string rewriting. By left-recursive, we mean that the recursive descent takes place only in the left partners. The definition of overlap closures recurses in both arguments (we always overlap a closure with a closure):

▶ **Definition A.1** ([10]). *For a rewrite system R, the set OC is defined as the least set such that*

1. $R \subseteq \mathit{OC}$,
2. *if* $(s, tx) \in \mathit{OC}$ *and* $(xu, v) \in \mathit{OC}$ *for some* $t, x, u \neq \epsilon$ *then* $(su, tv) \in \mathit{OC}$;
2'. *if* $(s, xt) \in \mathit{OC}$ *and* $(ux, v) \in \mathit{OC}$ *for some* $t, x, u \neq \epsilon$ *then* $(us, vt) \in \mathit{OC}$;
3 *if* $(s, tut') \in \mathit{OC}$ *and* $(u, v) \in \mathit{OC}$ *then* $(s, tvt') \in \mathit{OC}$;
3'. *if* $(u, v) \in \mathit{OC}$ *and* $(svs', t) \in \mathit{OC}$ *then* $(sus', t) \in \mathit{OC}$.

The following recursive definition is left-recursive (we overlap a closure with a rule). We need an extra rule (Item 4) and drop a rule (Item 3'), the others correspond to Definition A.1.

▶ **Definition A.2.** *For a rewrite system R, the set OC' is defined as the least set such that*

1. $R \subseteq \mathit{OC}'$,
2. *if* $(s, tx) \in \mathit{OC}'$ *and* $(xu, v) \in R$ *for some* $t, x, u \neq \epsilon$ *then* $(su, tv) \in \mathit{OC}'$;
2'. *if* $(s, xt) \in \mathit{OC}'$ *and* $(ux, v) \in R$ *for some* $t, x, u \neq \epsilon$ *then* $(us, vt) \in \mathit{OC}'$;
3. *if* $(s, tut') \in \mathit{OC}'$ *and* $(u, v) \in R$ *then* $(s, tvt') \in \mathit{OC}'$;
4. *if* $(s, tx) \in \mathit{OC}'$ *and* $(u, yv) \in \mathit{OC}'$ *and* $(xwy, z) \in R$ *for some* $t, x, y, v \neq \epsilon$ *then* $(swu, tzv) \in \mathit{OC}'$.

The main result of this Appendix is that the set OC' covers the overlap closures up to inverse rewriting of left hand sides:

▶ **Theorem A.3.** $\mathit{OC} = \{(s, t) \mid s \to_R^* s' \wedge (s', t) \in \mathit{OC}'\}$.

Since we are interested in right-hand sides of closures, these extra rewrite steps do not hurt.

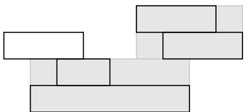
In order to prove Theorem A.3, it is useful to represent a closure by a tree that describes the way the closure is formed: the composition tree of the closure. Each node of a composition tree denotes an application of one of the inference rules of Definitions A.1 and A.2. An extra node type 3' denotes an $\to_R$-step as seen in Theorem A.3.

▶ **Definition A.4** ([7]). *Define the signature* $\Omega = \{1, 2, 2', 3, 3', 4\}$*, where* 1 *is unary,* 4 *is ternary, and the other symbols are binary. The set* CT *of* composition trees *is defined as the set of ground terms over* Ω*.*

▶ **Definition A.5.** *A composition tree represents a set of string pairs, as follows:*

$$\begin{aligned}
\langle 1 \rangle &= \{(\ell, r) \mid (\ell \to r) \in R\}, \\
\langle 2(c_1, c_2) \rangle &= \{(su, tv) \mid (s, tx) \in \langle c_1 \rangle, (xu, v) \in \langle c_2 \rangle, t, x, u \neq \epsilon\}, \\
\langle 2'(c_1, c_2) \rangle &= \{(us, vt) \mid (s, xt) \in \langle c_1 \rangle, (ux, v) \in \langle c_2 \rangle, t, x, u \neq \epsilon\}, \\
\langle 3(c_1, c_2) \rangle &= \{(s, tvt') \mid (s, tut') \in \langle c_1 \rangle, (u, v) \in \langle c_2 \rangle\}, \\
\langle 3'(c_1, c_2) \rangle &= \{(sus', t) \mid (svs', t) \in \langle c_1 \rangle, (u, v) \in \langle c_2 \rangle\}, \\
\langle 4(c_1, c_2, c_3) \rangle &= \{(swu, tzv) \mid (s, tx) \in \langle c_1 \rangle, (u, yv) \in \langle c_2 \rangle, \\
&\qquad (xwy, z) \in \langle c_3 \rangle, t, x, y, v \neq \epsilon\} \ .
\end{aligned}$$

▶ **Example A.6.** The composition tree $4(1, 2(1, 1), 3'(1, 1))$ denotes all pairs obtained by the following overlaps of rewrite steps. Times flows from top to bottom. Each of the rectangles of height 1 is a step, corresponding to a 1 node in the tree. The grey rectangle in the top right is $2(1, 1)$, the grey rectangle in the bottom is $3'(1, 1)$.



Let CT' denote the composition trees that do not contain the function symbol 4. By construction we have:

▶ **Lemma A.7.** $\mathit{OC} = \bigcup_{c \in \mathit{CT}'} \langle c \rangle$.

Adding symbols 4 does not increase expressiveness, since $\langle 4(c_1, c_2, c_3) \rangle \subseteq \langle 2(c_1, 2'(c_2, c_3)) \rangle$.

▶ **Lemma A.8.** $\mathit{OC} = \bigcup_{c \in \mathit{CT}} \langle c \rangle$.

In the remainder of this section, we give a semantics-preserving transformation from CT (arbitrary composition trees) to a subset that describes the right-hand side of Theorem A.3. Let us first characterize the goal precisely.

▶ **Definition A.9.** *The set* CT_N *is given by the regular tree grammar with variables* T, D *(top, deep), start variable* T*, and rules*

$$T \to 3'(1, T) \mid D, \qquad D \to 1 \mid 2(D, 1) \mid 2'(D, 1) \mid 3(D, 1) \mid 4(D, D, 1).$$

Rules for D correspond to the rules of Definition A.2, creating $(s', t) \in \mathsf{OC}'$. Rules for T correspond to the initial derivation $s \to_R^* s'$. Therefore,

▶ **Lemma A.10.** $\langle \mathit{CT}_N \rangle = \{(s, t) \mid s \to_R^* s' \wedge (s', t) \in \mathit{OC}'\}$.

We are going to construct a term rewriting system Q on Ω that has CT_N as normal forms. It must remove all non-1 symbols from the left argument of $3'$, and remove all non-1 symbols from the rightmost argument of $2, 2', 3$, and 4. Also, it must remove all $3'$ that are below some non-$3'$. These conditions already determine the set of left-hand sides of Q.

For each left-hand side l, the set of right-hand sides must cover l semantically:

$$\forall l \in \mathsf{lhs}(Q) : \langle l \rangle \subseteq \bigcup_{(l,r) \in Q} \langle r \rangle.$$

A term rewriting system Q over signature Ω with the desired properties is defined in Table 3. We bubble-up 3' symbols, e. g., $2(3'(c_1, c_2), c_3) \to 3'(c_1, 2(c_2, c_3))$ (Rule 9), and we rotate to move non-1 symbols, e. g., $2(c_1, 2(c_2, c_3)) \to 2(2(c_1, c_2), c_3)$ (Rule 1). Rotation below $3'$ goes to the left. Rules 3 and 13 show that symbol 4 cannot be avoided.

Table 3 The term rewriting system Q for composition trees.

Rule	No.	Rule	No.
$2(c_1, 2(c_2, c_3)) \to 2(2(c_1, c_2), c_3)$	(1)	$3(c_1, 4(c_2, c_2', c_3)) \to 3(3(3(c_1, c_2), c_2'), c_3)$	(29)
$2(c_1, 2(c_2, c_3)) \to 2(3(c_1, c_2), c_3)$	(2)	$3'(2(c_1, c_2), c_3) \to 3'(c_1, 3'(c_2, c_3))$	(30)
$2(c_1, 2'(c_2, c_3)) \to 4(c_1, c_2, c_3)$	(3)	$3'(2'(c_1, c_2), c_3) \to 3'(c_1, 3'(c_2, c_3))$	(31)
$2(c_1, 2'(c_2, c_3)) \to 3(2(c_1, c_2), c_3)$	(4)	$3'(3(c_1, c_2), c_3) \to 3'(c_1, 3'(c_2, c_3))$	(32)
$2(c_1, 3(c_2, c_3)) \to 3(2(c_1, c_2), c_3)$	(5)	$3'(3'(c_1, c_2), c_3) \to 3'(c_1, 3'(c_2, c_3))$	(33)
$2(c_1, 3'(c_2, c_3)) \to 3'(c_2, 2(c_1, c_3))$	(6)	$3'(4(c_1, c_1', c_2), c_3) \to 3'(c_1, 3'(c_1', 3'(c_2, c_3)))$	(34)
$2(c_1, 3'(c_2, c_3)) \to 2(2(c_1, c_2), c_3))$	(7)	$4(c_1, c_1', 2(c_2, c_3)) \to 4(2(c_1, c_2), c_1', c_3)$	(35)
$2(c_1, 3'(c_2, c_3)) \to 2(3(c_1, c_2), c_3))$	(8)	$4(c_1, c_1', 2(c_2, c_3)) \to 3(4(c_1, c_1', c_2), c_3)$	(36)
$2(3'(c_1, c_2), c_3) \to 3'(c_1, 2(c_2, c_3))$	(9)	$4(c_1, c_1', 2(c_2, c_3)) \to 4(3(c_1, c_2), c_1', c_3)$	(37)
$2(c_1, 4(c_2, c_2', c_3)) \to 4(2(c_1, c_2), c_2', c_3)$	(10)	$4(c_1, c_1', 2'(c_2, c_3)) \to 3(4(c_1, c_1', c_2), c_3)$	(38)
$2(c_1, 4(c_2, c_2', c_3)) \to 4(3(c_1, c_2), c_2', c_3)$	(11)	$4(c_1, c_1', 2'(c_2, c_3)) \to 4(c_1, 2(c_1', c_2), c_3)$	(39)
$2(c_1, 4(c_2, c_2', c_3)) \to 3(3(2(c_1, c_2'), c_2), c_3)$	(12)	$4(c_1, c_1', 2'(c_2, c_3)) \to 4(c_1, 3(c_1', c_2), c_3)$	(40)
$2'(c_1, 2(c_2, c_3)) \to 4(c_1, c_2, c_3)$	(13)	$4(c_1, c_1', 3(c_2, c_3)) \to 3(4(c_1, c_1', c_2), c_3)$	(41)
$2'(c_1, 2(c_2, c_3)) \to 3(2'(c_1, c_2), c_3)$	(14)	$4(c_1, c_1', 3'(c_2, c_3)) \to 3'(c_2, 4(c_1, c_1', c_3))$	(42)
$2'(c_1, 2'(c_2, c_3)) \to 2'(2'(c_1, c_2), c_3)$	(15)	$4(c_1, c_1', 3'(c_2, c_3)) \to 4(2(c_1, c_2), c_1', c_3)$	(43)
$2'(c_1, 2'(c_2, c_3)) \to 2'(3(c_1, c_2), c_3)$	(16)	$4(c_1, c_1', 3'(c_2, c_3)) \to 4(c_1, 2'(c_1', c_2), c_3)$	(44)
$2'(c_1, 3(c_2, c_3)) \to 3(2'(c_1, c_2), c_3)$	(17)	$4(c_1, c_1', 3'(c_2, c_3)) \to 3(4(c_1, c_1', c_2), c_3)$	(45)
$2'(c_1, 3'(c_2, c_3)) \to 3'(c_2, 2'(c_1, c_3))$	(18)	$4(c_1, c_1', 3'(c_2, c_3)) \to 4(3(c_1, c_2), c_1', c_3)$	(46)
$2'(c_1, 3'(c_2, c_3)) \to 2'(2'(c_1, c_2), c_3)$	(19)	$4(c_1, c_1', 3'(c_2, c_3)) \to 4(c_1, 3(c_1', c_2), c_3)$	(47)
$2'(c_1, 4(c_2, c_2', c_3)) \to 4(c_2, 2'(c_1, c_2'), c_3)$	(20)	$4(3'(c_1, c_2), c_1', c_3) \to 3'(c_1, 4(c_2, c_1', c_3))$	(48)
$2'(c_1, 4(c_2, c_2', c_3)) \to 4(c_2, 3(c_1, c_2'), c_3)$	(21)	$4(c_1, 3'(c_1', c_2), c_3) \to 3'(c_1', 4(c_1, c_2, c_3))$	(49)
$2'(c_1, 4(c_2, c_2', c_3)) \to 3(3(2'(c_1, c_2), c_2'), c_3)$	(22)	$4(c_1, c_1', 4(c_2, c_2', c_3)) \to 4(2(c_1, c_2), 2'(c_1', c_2'), c_3)$	(50)
$2'(3'(c_1, c_2), c_3) \to 3'(c_1, 2'(c_2, c_3)),$	(23)	$4(c_1, c_1', 4(c_2, c_2', c_3)) \to 4(3(c_1, c_2), 2'(c_1', c_2'), c_3)$	(51)
$3(c_1, 2(c_2, c_3)) \to 3(3(c_1, c_2), c_3)$	(24)	$4(c_1, c_1', 4(c_2, c_2', c_3)) \to 4(2(c_1, c_2), 3(c_1', c_2'), c_3)$	(52)
$3(c_1, 2'(c_2, c_3)) \to 3(3(c_1, c_2), c_3)$	(25)	$4(c_1, c_1', 4(c_2, c_2', c_3)) \to 4(3(c_1, c_2), 3(c_1', c_2'), c_3)$	(53)
$3(c_1, 3(c_2, c_3)) \to 3(3(c_1, c_2), c_3)$	(26)	$4(c_1, c_1', 4(c_2, c_2', c_3)) \to 3(3(4(c_1, c_1', c_2'), c_2), c_3)$	(54)
$3(c_1, 3'(c_2, c_3)) \to 3(3(c_1, c_2), c_3)$	(27)	$4(c_1, c_1', 4(c_2, c_2', c_3)) \to 3(3(4(c_1, c_1', c_2), c_2'), c_3)$	(55)
$3(3'(c_1, c_2), c_3) \to 3'(c_1, 3(c_2, c_3))$	(28)		

Termination of Q follows from a lexicographic combination of an interpretation ρ that decreases under rotation, and an interpretation σ that decreases under bubbling.

▶ **Lemma A.11.** *Q terminates.*

Proof. Let the two interpretations ρ and σ on natural numbers be defined by

$$\begin{aligned} \rho(1) &= 2, \\ \rho(2(c_1, c_2)) = \rho(2'(c_1, c_2)) = \rho(3(c_1, c_2)) &= \rho(c_1) + 2\rho(c_2), \\ \rho(3'(c_1, c_2)) &= 2\rho(c_1) + \rho(c_2), \\ \rho(4(c_1, c_2, c_3)) &= \rho(c_1) + \rho(c_2) + 2\rho(c_3), \end{aligned}$$

$$\begin{aligned}\sigma(1) &= 2,\\ \sigma(2(c_1,c_2)) &= \sigma(2'(c_1,c_2)) = \sigma(3(c_1,c_2)) = \sigma(c_1)\cdot\sigma(c_2),\\ \sigma(3'(c_1,c_2)) &= \sigma(c_1)\cdot\sigma(c_2)+1,\\ \sigma(4(c_1,c_2,c_3)) &= \sigma(c_1)\cdot\sigma(c_2)\cdot\sigma(c_3)\ .\end{aligned}$$

The order $>$ on terms defined by $s > t$ if $\rho(s) > \rho(t)$ or $\rho(s) = \rho(t)$ and $\sigma(s) > \sigma(t)$ is a reduction order. With this, the rules $\ell \to r$ in 9, 28, 48, and 49 satisfy $\rho(\ell) = \rho(r)$ and $\sigma(\ell) > \sigma(r)$. For instance, Rule 49 satisfies $\rho(\ell) = \rho(r) = \rho(c_1) + \rho(c_1') + \rho(c_2) + 2\rho(c_3)$ and $\sigma(\ell) = (\sigma(c_1)\sigma(c_2)+1)\sigma(c_1')\sigma(c_3) > \sigma(c_1)\sigma(c_2)\sigma(c_1')\sigma(c_3)+1 = \sigma(r)$. All other rules $\ell \to r$ in Q satisfy $\rho(\ell) > \rho(r)$. For instance, Rule 54 satisfies $\rho(\ell) = \rho(c_1) + \rho(c_1') + 2\rho(c_2) + 2\rho(c_2') + 4\rho(c_3) > \rho(c_1) + \rho(c_1') + 2\rho(c_2) + 2\rho(c_2') + 2\rho(c_3) = \rho(r)$. So Q is ordered by the reduction order $>$, and so Q terminates. ◀

▶ **Lemma A.12.** *For every composition tree c that admits a Q rewrite step, and for every $(s,t) \in \langle c\rangle$ there is a composition tree c' such that both $c \to_Q c'$ and $(s,t) \in \langle c'\rangle$.*

Proof. The proof is done by a case analysis over all left hand sides of Q. We show only one particularly complex case; the other cases work similarly.

Let $c = 4(c_1, c_1', 4(c_2, c_2', c_3))$. By definition of $\langle\cdot\rangle$, we get $s = \hat{s}wu$, $t = \hat{t}zv$, $(\hat{s}, \hat{t}x) \in \langle c_1\rangle$, $(u, yv) \in \langle c_1'\rangle$, $(xwy, z) \in \langle 4(c_2, c_2', c_3)\rangle$ for some $\hat{t}, x, y, v \neq \epsilon$. Again, we get $xwy = s'w'u'$, $z = t'z'v'$, $(s', t'x') \in \langle c_2\rangle$, $(u', y'v') \in \langle c_2'\rangle$, $(x'w'y', z') \in \langle c_3\rangle$ for some $t', x', y', v' \neq \epsilon$. We distinguish cases according to the overlaps:

1. $x \in \mathsf{Prefix}(s')$, $y \in \mathsf{Suffix}(u')$. Then $(\hat{s}s'', \hat{t}t'x') \in \langle 2(c_1,c_2)\rangle$ where $s'' \neq \epsilon$ is defined by $s' = xs''$. Next, $(u''u, y'v'v) \in \langle 2'(c_1', c_2')\rangle$ where $u'' \neq \epsilon$ is defined by $u' = u''y$. Finally, $(s,t) = (\hat{s}s''w'u''u, \hat{t}t'z'v'v) \in \langle 4(2(c_1,c_2), 2'(c_1',c_2'), c_3)\rangle$, and we choose $c \to c' = 4(2(c_1,c_2), 2'(c_1',c_2'), c_3)$ by Rule 50.
2. s' is a prefix of x, $x \in \mathsf{Prefix}(s'w')$, $y \in \mathsf{Suffix}(u')$. Then $(\hat{s}, \hat{t}t'x'') \in \langle 3(c_1,c_2)\rangle$ where x'' is defined by $x = s'x''$. Next, $(u''u, y'v'v) \in \langle 2'(c_1',c_2')\rangle$ where $u'' \neq \epsilon$ is defined by $u' = u''y$. Finally, $(s,t) = (\hat{s}w''u''u, \hat{t}t'z'v'v) \in \langle 4(3(c_1,c_2), 2'(c_1',c_2'), c_3)\rangle$, where w'' is defined by $w' = x''w''$, and we choose $c \to c' = 4(3(c_1,c_2), 2'(c_1',c_2'), c_3)$ by Rule 51.
3. $x \in \mathsf{Prefix}(s')$, u' is a suffix of y, $y \in \mathsf{Suffix}(w'u')$. This case is symmetric to Case 2. We use Rule 52.
4. s' is a prefix of x, u' is a suffix of y. Then $(\hat{s}, \hat{t}t'x'') \in \langle 3(c_1,c_2)\rangle$ where x'' is defined by $x = s'x''$. Next, $(u, y''v'v) \in \langle 3(c_1',c_2')\rangle$ where y'' is defined by $y = y''u'$. Finally, $(s,t) = (\hat{s}w''u, \hat{t}t'z'v'v) \in \langle 4(3(c_1,c_2), 3(c_1',c_2'), c_3)\rangle$, where w'' is defined by $w' = x''w''y''$, and we choose $c \to c' = 4(3(c_1,c_2), 3(c_1',c_2'), c_3)$ by Rule 53.
5. $s'w'$ is a prefix of x, $y \in \mathsf{Suffix}(u')$. Then $(\hat{s}u''u, \hat{t}y'v') \in \langle 4(c_1,c_2,c_2')\rangle$ where x'' is defined by $x = s'w'x''$, and u'' is defined by $u' = x''u''$. Next, $(\hat{s}u''u, \hat{t}t'x'w'y'v'v) \in \langle 3(4(c_1,c_2,c_2'), c_1')\rangle$. Finally, $(s,t) = (\hat{s}u''u, \hat{t}t'z'v'v) \in \langle 3(3(4(c_1,c_2,c_2'),c_1'),c_3)\rangle$, by Rule 54.
6. $x \in \mathsf{Prefix}(s')$, $w'u'$ is a suffix of y. This case is symmetric to Case 5. We use Rule 55. ◀

▶ **Lemma A.13.** *For every composition tree c that is in Q-normal form and does not contain any $3'$ symbols, we have $\langle c\rangle \subseteq \mathsf{OC}'$.*

Proof. The claim is proven by induction on $|c|$ as follows. If $c = 1$ then $\langle c\rangle = R \subseteq \mathsf{OC}'$. If $c = 2(c_1,c_2)$ then $c_2 = 1$ because c is in Q-normal form. From the inductive hypothesis for c_1 we get $\langle c_1\rangle \subseteq \mathsf{OC}'$; so $\langle c\rangle \subseteq \mathsf{OC}'$. The same argument applies when $c = 2'(c_1,c_2)$ or $c = 3(c_1,c_2)$. If $c = 4(c_1,c_2,c_3)$ then $c_3 = 1$ because c is in Q-normal form. From the inductive hypothesis for c_1 we get $\langle c_1\rangle \subseteq \mathsf{OC}'$. From the inductive hypothesis for c_2 we get $\langle c_2\rangle \subseteq \mathsf{OC}'$. So $\langle c\rangle \subseteq \mathsf{OC}'$. ◀

Now we are ready to prove Theorem A.3.

Proof of Theorem A.3. We prove that $c \in \mathsf{OC}$ and $(s,t) \in \langle c \rangle$ implies $s \to_R^* s'$ and $(s',t) \in \mathsf{OC}'$ for some s'. We do so by induction on c, ordered by $>$. If c admits a Q rewrite step then by Lemma A.12 there is a composition tree c' such that both $c \to_Q c'$ and $(s,t) \in \langle c' \rangle$. Because $c > c'$, the claim follows by inductive hypothesis for c'. Now suppose that c is in Q-normal form. If c does not contain any $3'$ symbol then $(s,t) \in \mathsf{OC}'$ by Lemma A.13, and we choose $s' = s$. Else, because c is in Q-normal form, $c = 3'(1, c_2)$ for some c_2. Let $(s'',t) \in \langle c_2 \rangle$ and $s \to_R s''$. From the inductive hypothesis for c_2 we get $s'' \to_R^* s'$ and $(s',t) \in \mathsf{OC}'$ for some s'. So $s \to_R s'' \to_R^* s'$ and the claim holds. ◀

From Theorem A.3, we immediately get:

▶ **Corollary A.14.** *rhs*$(OC) =$ *rhs*(OC').

Because OC' is left-recursive, we can derive a recursive characterization of the set of right hand sides of overlap closures:

▶ **Corollary A.15** (This is Corollary 7.1)**.** *rhs*(OC) *is the least set S such that*

1. *rhs*$(R) \subseteq S$*,*
2. *if $tx \in S$ and $(xu, v) \in R$ for some $t, x, u \neq \epsilon$ then $tv \in S$;*
3. *if $xt \in S$ and $(ux, v) \in R$ for some $t, x, u \neq \epsilon$ then $vt \in S$;*
4. *if $tut' \in S$ and $(u, v) \in R$ then $tvt' \in S$;*
5. *if $tx \in S$ and $yv \in S$ and $(xwy, z) \in R$ for some $t, x, y, v \neq \epsilon$ then $tzv \in S$.*

Proof Nets for First-Order Additive Linear Logic

Willem B. Heijltjes
University of Bath, United Kingdom
http://willem.heijltj.es

Dominic J. D. Hughes
Logic Group, UC Berkeley, USA
http://boole.stanford.edu/~dominic

Lutz Straßburger
Inria Saclay, Palaiseau, France
LIX, École Polytechnique, Palaiseau, France
http://www.lix.polytechnique.fr/Labo/Lutz.Strassburger

Abstract

We present canonical proof nets for first-order additive linear logic, the fragment of linear logic with sum, product, and first-order universal and existential quantification. We present two versions of our proof nets. One, *witness nets*, retains explicit witnessing information to existential quantification. For the other, *unification nets*, this information is absent but can be reconstructed through unification. Unification nets embody a central contribution of the paper: first-order witness information can be left implicit, and reconstructed as needed. Witness nets are canonical for first-order additive sequent calculus. Unification nets in addition factor out any inessential choice for existential witnesses. Both notions of proof net are defined through coalescence, an additive counterpart to multiplicative contractibility, and for witness nets an additional geometric correctness criterion is provided. Both capture sequent calculus cut-elimination as a one-step global composition operation.

2012 ACM Subject Classification Theory of computation → Proof theory; Theory of computation → Linear logic

Keywords and phrases linear logic, first-order logic, proof nets, Herbrand's theorem

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.22

Related Version A full version [13] is available at http://hal.inria.fr/hal-01867625/.

Funding *Willem B. Heijltjes*: was supported by EPSRC Project EP/R029121/1 *Typed Lambda-Calculi with Sharing and Unsharing.*
Lutz Straßburger: was supported by the ANR-FWF international grant ANR-5-CE25-0014 *The Fine Structure of Formal Proof Systems and their Computational Interpretations (FISP).*

Acknowledgements We would like to thank the anonymous referees for their constructive feedback. Dominic Hughes thanks his hosts, Wes Holliday and Dana Scott, at the UC Berkeley Logic Group.

1 Introduction

Additive linear logic (ALL) is the logic of sum, product, and their canonical morphisms: projections, injections, and diagonals. Semantically, the logic represents parallel communication between two parties, with sum and product as respectively the sending and receiving of a binary choice [22, 3]. As such it is a core part of *session types* for process calculi [16, 2, 28].

A microcosm of parallellism, ALL already demonstrates the *Blass problem* of game semantics [1], that sequential strategies do not in general have associative composition. This is resolved by *proof nets* [7, 21], which are a canonical, *true-concurrency* presentation of ALL.

Here, we extend proof nets to first-order additive linear logic (ALL1). Beyond the solution to the proof-net problem, a main contribution is the (further) development of the two techniques we consider: *explicit substitutions* for witness assignment, and reconstruction of

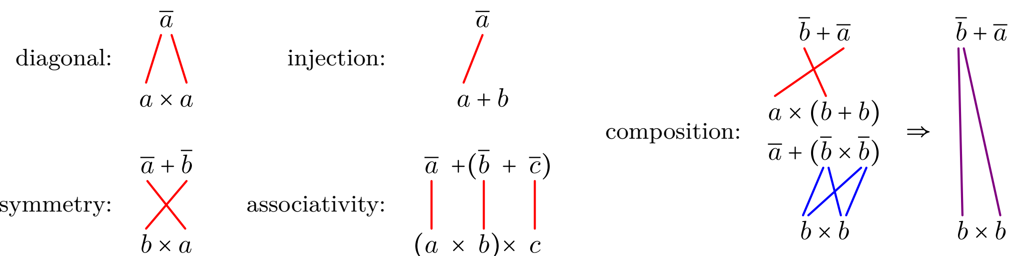
© Willem B. Heijltjes, Dominic J. D. Hughes, and Lutz Straßburger;
licensed under Creative Commons License CC-BY
4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019).
Editor: Herman Geuvers; Article No. 22; pp. 22:1–22:22
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

witness information through *unification*, as pioneered for MLL by the second author [18]. We expect to apply these to first-order logics more generally.

Proofs have been relegated to the appendix – for a full version see the report [13].

Additive proof nets

ALL proof nets [21, Section 4.10] represent a morphism $A \to B$ by a sequent $\vdash \overline{A}, B$ plus a **linking**, a relation between the propositional atoms of $\overline{A}$ (the dual of A) and those of B. They are **canonical**: they factor out the permutations of sequent calculus, and correspond 1–1 to morphisms of the free category with binary sums and products. **Composition**, of proof nets over $\vdash \overline{A}, B$ and $\vdash \overline{B}, C$ to one over $\vdash \overline{A}, C$, is by the relational composition of their linkings along the dual formulas B and $\overline{B}$ and captures sequent-calculus cut-elimination. Below are examples of proof nets and their composition.



We extend additive proof nets with first-order quantification. Our central challenge is to incorporate the essential content of first-order proof, the **witness assignment** to existential quantifiers. Commonly, as in the sequent calculus rule below left, a witness to $\exists x.B$ is given by an immediate substitution $B[t/x]$. To assign different witnesses in different branches of a proof, the subformula is duplicated first, giving $B[s/x]$ and $B[t/x]$, as below right.

$$\frac{\vdash A, B[t/x]}{\vdash A, \exists x.B}\ \exists\mathrm{R},t \qquad \frac{\dfrac{\overline{\vdash \overline{P}(s), P(s)}}{\vdash \exists x.\overline{P}(x), P(s)}\ \exists\mathrm{R},s \qquad \dfrac{\overline{\vdash \overline{P}(t), P(t)}}{\vdash \exists x.\overline{P}(x), P(t)}\ \exists\mathrm{R},t}{\vdash \exists x.\overline{P}(x), P(s) \times P(t)}\ \times\mathrm{R}$$

This is incompatible with a **sequent + links** proof net design, where the conclusion sequent remains intact, and a subformula B cannot be the subject of substitution or duplication. Instead, we propose two alternative treatments of witnessing terms, embodied in two notions of proof net: **witness nets** and **unification nets**. Our solutions are based on the second author's recent **unification nets** for first-order multiplicative linear logic [18]. Their main feature is to omit existential witnesses altogether, and reconstruct them by unification.

Witness nets record witness assignment in substitution maps attached to each link. The example below left shows the proof net for the sequent proof earlier. (We will assume a different variable for each quantifier, and we attach links to predicate letters, as the root connective of an atomic proposition.) Witness nets are canonical for ALL1 sequent calculus permutations. Composition is direct, where the witness assignments of links are composed through a simple process of *interaction + hiding* similar to that of game semantics [26].

Unification nets omit any witness information, as illustrated below right. In addition to canonicity, they embody a notion of **generality**: where more than one witness could be assigned, unification nets do not require a definite choice, while witness nets do. Composition is direct, by relational composition. We compare further properties in Figure 11 in the conclusion, where we also discuss related work and Lambek's notion of **generality** [23].

a witness net: $[s/x]$ $[t/x]$ — $\exists x.\overline{P}(x)$ linked to $P(s) \times P(t)$

a unification net: $\exists x.\overline{P}(x)$ linked to $P(s) \times P(t)$

Background

Additive linear logic is combinatorially rich, yet well-behaved and tractable: proof search [6] and proof net correctness [12] for a net over $\vdash A, B$ are linear in $|A| \times |B|$ (with $|A|$ the size of the syntax tree of A). Proof nets remain canonical and equally tractable when extended with the two units [11, 12], and the first-order case is merely NP-complete [12].

ALL is of course part of MALL (multiplicative-additive linear logic), and its lessons are clearly visible in the second author's canonical MALL proof nets [21], as well as the first and second author's locally canonical *conflict nets* [20]. Its proof nets also appear in the third author's study of the *medial rule* for classical logic [27], and as the *skew fibrations* in the second author's *combinatorial proofs* for classical logic [17]. To prepare the ground for cut elimination in first-order combinatorial proofs [19] is further motivation for the present work.

Proof identity

At the heart of a theory of proof nets is the question of *proof identity*: when are two proofs equivalent? The answer determines which proofs should map onto the same proof net. The introduction of quantifiers creates an interesting issue: if two proofs differ by an immaterial choice of existential witness, should they be equivalent? For example, to prove the sequent $\vdash \exists x.P(x), \exists y.\overline{P}(y)$ both quantifiers must receive *the same* witness, as in the following two proofs, but *any* witness will do.

$$\dfrac{\dfrac{}{\vdash P(s), \overline{P}(s)}}{\vdash \exists x.P(x), \exists y.\overline{P}(y)} \quad \stackrel{?}{\equiv} \quad \dfrac{\dfrac{}{\vdash P(t), \overline{P}(t)}}{\vdash \exists x.P(x), \exists y.\overline{P}(y)}$$

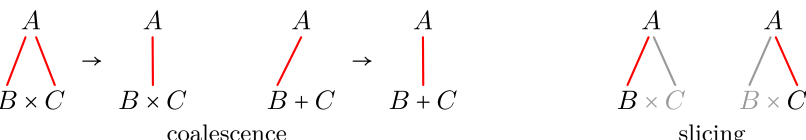
The issue is more pronounced where quantifiers are *vacuous*, $\exists x.A$ with x not free in A. The proofs below left can only be distinguished even syntactically because the $\exists$R rule makes the instantiating witness explicit. Below right is an interesting intermediate variant: the witness s or t can be observed without explicit annotation in the $\exists$R rule, but the choice is equally immaterial to the content of the proof as when the quantifier were vacuous.

$$\dfrac{\dfrac{}{\vdash P, \overline{P}}}{\vdash \exists x.P, \overline{P}}\,\exists\text{R},s \quad \stackrel{?}{\equiv} \quad \dfrac{\dfrac{}{\vdash P, \overline{P}}}{\vdash \exists x.P, \overline{P}}\,\exists\text{R},t \qquad \dfrac{\dfrac{\dfrac{}{\vdash P, \overline{P}}}{\vdash P + Q(s), \overline{P}}\,+\text{R},1}{\vdash \exists x.P + Q(x), \overline{P}}\,\exists\text{R},s \quad \stackrel{?}{\equiv} \quad \dfrac{\dfrac{\dfrac{}{\vdash P, \overline{P}}}{\vdash P + Q(t), \overline{P}}\,+\text{R},1}{\vdash \exists x.P + Q(x), \overline{P}}\,\exists\text{R},t$$

In this paper we will not attempt to settle the question of proof identity. Rather, our two notions of proof net each represent a natural and coherent perspective, at either end of the spectrum. *Witness nets* make all existential witnesses explicit, including those to vacuous quantifiers, rejecting all three equivalences above. *Unification nets* leave all witnesses implicit, thus identifying all proofs modulo witness assignment, and validating all three equivalences.

Correctness: coalescence and slicing

Additive proof nets have two natural correctness criteria. **Coalescence** [12, 20], a counterpart to multiplicative *contractibility* [4, 9], provides efficient correctness and sequentialization via local rewriting: it asks that the steps below left result in a single link, connecting both formulas. **Slicing** [21] is a global, geometric criterion: it asks that each **slice**, a choice to remove one subformula of each product along with all connected links, retains a single link.



We extend coalescence to both witness nets and unification nets, and slicing to witness nets. Here, we illustrate the former, and leave a discussion of slicing to the conclusion of the paper.

We will distinguish **strict coalescence** (→) for witness nets and **unifying coalescence** ($\rightsquigarrow$) for unification nets. We first give an example of the former (abbreviating $P(x,y)$ to Pxy). In the initial witness net, below left, each link corresponds to a sequent calculus axiom between both linked subformulas, with the substitutions applied.

$$\forall x.\exists y.\overline{P}xy \;\; [x/z,s/y] \;\; [x/z,t/y] \;\; \exists z.Pzs \times Pzt \;\rightarrow\; \forall x.\exists y.\overline{P}xy \;\; [x/z] \;\; [x/z,t/y] \;\; \exists z.Pzs \times Pzt \;\rightarrow\; \forall x.\exists y.\overline{P}xy \;\; [x/z] \;\; [x/z] \;\; \exists z.Pzs \times Pzt \;\rightarrow \cdots$$

$$\dfrac{}{\vdash \overline{P}xs, Pxs} \quad \dfrac{}{\vdash \overline{P}xt, Pxt} \qquad\qquad \dfrac{\dfrac{}{\vdash \overline{P}xs, Pxs}}{\vdash \exists y.\overline{P}xy, Pxs}\,\exists\text{R},s \quad \dfrac{}{\vdash \overline{P}xt, Pxt} \qquad\qquad \dfrac{\dfrac{}{\vdash \overline{P}xs, Pxs}}{\vdash \exists y.\overline{P}xy, Pxs}\,\exists\text{R},s \quad \dfrac{\dfrac{}{\vdash \overline{P}xt, Pxt}}{\vdash \exists y.\overline{P}xy, Pxt}\,\exists\text{R},t$$

The first two steps (above middle and right) move both links from the subformula $\overline{P}xy$ to $\exists y.\overline{P}xy$, removing the substitutions $[s/y]$ and $[t/y]$ on y, and corresponding to sequent rules $\exists\text{R}, s$ and $\exists\text{R}, t$. Observe that we maintain the domain of the substitutions on a link as the variables of those existential quantifiers that either linked subformula is (strictly) in scope of.

The next step, from above right to below left, combines both links, and corresponds to an additive conjunction rule. We require that both substitutions agree (their domains are the same by the above observation); hence this step could not have been performed before the previous two, corresponding to the non-permutability of the generated inference rules.

$$\cdots \rightarrow \;\; \forall x.\exists y.\overline{P}xy \;\; [x/z] \;\; \exists z.Pzs \times Pzt \;\rightarrow\; \forall x.\exists y.\overline{P}xy \;\; \exists z.Pzs \times Pzt \;\rightarrow\; \forall x.\exists y.\overline{P}xy \;\; \exists z.Pzs \times Pzt$$

$$\dfrac{\dfrac{\dfrac{}{\vdash \overline{P}xs, Pxs}}{\vdash \exists y.\overline{P}xy, Pxs} \quad \dfrac{\dfrac{}{\vdash \overline{P}xt, Pxt}}{\vdash \exists y.\overline{P}xy, Pxt}}{\vdash \exists y.\overline{P}xy, Pxs \times Pxt}\,\times\text{R} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \overline{P}xs, Pxs}}{\vdash \exists y.\overline{P}xy, Pxs} \quad \dfrac{\dfrac{}{\vdash \overline{P}xt, Pxt}}{\vdash \exists y.\overline{P}xy, Pxt}}{\vdash \exists y.\overline{P}xy, Pxs \times Pxt}}{\vdash \exists y.\overline{P}xy, \exists z.Pzs \times Pzt}\,\exists\text{R},x \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \overline{P}xs, Pxs}}{\vdash \exists y.\overline{P}xy, Pxs} \quad \dfrac{\dfrac{}{\vdash \overline{P}xt, Pxt}}{\vdash \exists y.\overline{P}xy, Pxt}}{\vdash \exists y.\overline{P}xy, Pxs \times Pxt}}{\vdash \exists y.\overline{P}xy, \exists z.Pzs \times Pzt}}{\vdash \forall x.\exists y.\overline{P}xy, \exists z.Pzs \times Pzt}\,\forall\text{R}$$

The final steps introduce an inference $\exists R, x$ and one $\forall R$. For the latter, we require that the *eigenvariable* x of the universal quantification does not occur in the range of the substitution of the link, as in $[x/z]$ in the net above left – hence the two steps could not be interchanged. It corresponds to the *eigenvariable condition* on the $\forall R$ rule that x is not free in the context.

Unifying coalescence, for unification nets, is similar to strict coalescence; two differences allow it to reconstruct witnesses by unification, which we illustrate. To initialize coalescence, the links in the net below left are given, as a substitution map, the **most general unifier** of the two propositions connected by the link. Both links now correspond to sequent axioms.

$\overline{P}xx$ / $Pys \times Ptz$ $\rightsquigarrow$ $\overline{P}xx$ [s/y,s/x] [t/z,t/x] $Pys \times Ptz$ $\rightsquigarrow$ $\overline{P}xx$ [u/z,u/y,u/x] $Pys \times Ptz$

$$\frac{}{\vdash \overline{P}ss, Pss} \qquad \frac{}{\vdash \overline{P}tt, Ptt} \qquad \frac{\overline{\vdash \overline{P}uu, Puu} \quad \overline{\vdash \overline{P}uu, Puu}}{\vdash \overline{P}uu, Puu \times Puu}\times\mathrm{R}$$

The second difference is the coalescence step for additive conjunction, above right. Where strict coalescence requires both links to carry identical substitution maps, here we require both maps to be *unifyable*, in the sense that both must have a common, more special (less general) substitution map. This is then applied to the new link. In the example, the terms s and t are unified to u, i.e. u is a most general term that specializes both s and t. Note that in the sequentialization, both subproofs also need to be specialized, from s and t to u.

2 Proof nets for first-order additive linear logic

First-order terms and the formulas of first-order ALL are generated by the following grammars.

$$t ::= x \mid f(t_1, \dots, t_n)$$

$$a ::= P(t_1, \dots, t_n) \mid \overline{P}(t_1, \dots, t_n)$$

$$A ::= a \mid A + A \mid A \times A \mid \exists x.A \mid \forall x.A$$

Negation $(\overline{\cdot})$ is applied to predicate symbols, $\overline{P}$ as a matter of convenience. The **dual** $\overline{A}$ of an arbitrary formula A is given by De Morgan. We use the following notational conventions:

x, y, z	$\in$ VAR	first-order variables
f, g, h	$\in \Sigma_f$	n-ary $(n \geq 0)$ function symbols from a fixed alphabet Σ_f
P, Q, R	$\in \Sigma_p$	n-ary $(n \geq 0)$ predicate symbols from a fixed alphabet Σ_p
s, t, u	$\in$ TERM	first-order terms over VAR and Σ_f
a, b, c	$\in$ ATOM	atomic propositions
A, B, C	$\in$ FORM	ALL1 formulas

A **sequent** $\vdash A, B$ is a pair of formulas A and B. A sequent calculus for ALL1 is given in Figure 1, where each rule has a symmetric counterpart for the first formula in the sequent. We write $\pi \vdash A, B$ for a proof π with conclusion sequent $\vdash A, B$. Two proofs are **equivalent** $\pi \sim \pi'$ if one is obtained from the other by rule permutations, given in Figure 2.

By a **subformula** we will mean a subformula **occurrence**. For instance, a formula $A \times A$ has two subformulas A, one on the left and one on the right. The **subformulas** $\mathrm{SUB}(A)$ of a formula are defined as follows; we write $B \leq A$ if B is a subformula of A, i.e. if $B \in \mathrm{SUB}(A)$.

$$\mathrm{SUB}(A) = \{A\} \cup \begin{cases} \mathrm{SUB}(B) \uplus \mathrm{SUB}(C) & \text{if } A = B + C \text{ or } A = B \times C \\ \mathrm{SUB}(B) & \text{if } A = \exists x.B \text{ or } A = \forall x.B \end{cases}$$

$$\frac{}{\vdash a, \overline{a}}\,\text{ax} \qquad \frac{\vdash A, B_i}{\vdash A, B_1 + B_2}\,{+}\text{R},i \qquad \frac{\vdash A, B \quad \vdash A, C}{\vdash A, B \times C}\,\times\text{R} \qquad \frac{\vdash A, B[t/x]}{\vdash A, \exists x.B}\,\exists\text{R},t \qquad \frac{\vdash A, B}{\vdash A, \forall x.B}\,\forall\text{R}\ (x \notin \text{FV}(A))$$

Figure 1 A sequent calculus for ALL1.

$$\frac{\dfrac{\vdash A, B}{\vdash A, \forall y.B}\,\forall\text{R}}{\vdash \forall x.A, \forall y.B}\,\forall\text{R} \quad\sim\quad \frac{\dfrac{\vdash A, B}{\vdash \forall x.A, B}\,\forall\text{R}}{\vdash \forall x.A, \forall y.B}\,\forall\text{R}$$

$$\frac{\dfrac{\vdash A, B[t/y]}{\vdash A, \exists y.B}\,\exists\text{R},t}{\vdash \forall x.A, \exists y.B}\,\forall\text{R} \quad\sim\quad \frac{\dfrac{\vdash A, B[t/y]}{\vdash \forall x.A, B[t/y]}\,\forall\text{R}}{\vdash \forall x.A, \exists y.B}\,\exists\text{R},t$$

$$\frac{\dfrac{\vdash A, B_i}{\vdash A, B_1{+}B_2}\,{+}\text{R},i}{\vdash \forall x.A, B_1{+}B_2}\,\forall\text{R} \quad\sim\quad \frac{\dfrac{\vdash A, B_i}{\vdash \forall x.A, B_i}\,\forall\text{R}}{\vdash \forall x.A, B_1{+}B_2}\,{+}\text{R},i$$

$$\frac{\dfrac{\vdash A, B \quad \vdash A, C}{\vdash A, B{\times}C}\,\times\text{R}}{\vdash \forall x.A, B{\times}C}\,\forall\text{R} \quad\sim\quad \frac{\dfrac{\vdash A, B}{\vdash \forall x.A, B}\,\forall\text{R} \quad \dfrac{\vdash A, C}{\vdash \forall x.A, C}\,\forall\text{R}}{\vdash \forall x.A, B{\times}C}\,\times\text{R}$$

$$\frac{\dfrac{\vdash A[s/x], B[t/y]}{\vdash A[s/x], \exists y.B}\,\exists\text{R},t}{\vdash \exists x.A, \exists y.B}\,\exists\text{R},s \quad\sim\quad \frac{\dfrac{\vdash A[s/x], B[t/y]}{\vdash \exists x.A, B[t/y]}\,\exists\text{R},s}{\vdash \exists x.A, \exists y.B}\,\exists\text{R},t$$

$$\frac{\dfrac{\vdash A[t/x], B_i}{\vdash A[t/x], B_1{+}B_2}\,{+}\text{R},i}{\vdash \exists x.A, B_1{+}B_2}\,\exists\text{R},t \quad\sim\quad \frac{\dfrac{\vdash A[t/x], B_i}{\vdash \exists x.A, B_i}\,\exists\text{R},t}{\vdash \exists x.A, B_1{+}B_2}\,{+}\text{R},i$$

$$\frac{\dfrac{\vdash A[t/x], B \quad \vdash A[t/x], C}{\vdash A[t/x], B{\times}C}\,\times\text{R}}{\vdash \exists x.A, B{\times}C}\,\exists\text{R},t \quad\sim\quad \frac{\dfrac{\vdash A[t/x], B}{\vdash \exists x.A, B}\,\exists\text{R},t \quad \dfrac{\vdash A[t/x], C}{\vdash \exists x.A, C}\,\exists\text{R},t}{\vdash \exists x.A, B{\times}C}\,\times\text{R}$$

$$\frac{\dfrac{\vdash A_i, B_j}{\vdash A_i, B_1{+}B_2}\,{+}\text{R},j}{\vdash A_1{+}A_2, B_1{+}B_2}\,{+}\text{R},i \quad\sim\quad \frac{\dfrac{\vdash A_i, B_j}{\vdash A_1{+}A_2, B_j}\,{+}\text{R},i}{\vdash A_1{+}A_2, B_1{+}B_2}\,{+}\text{R},j$$

$$\frac{\dfrac{\vdash A_i, B \quad \vdash A_i, C}{\vdash A_i, B{\times}C}\,\times\text{R}}{\vdash A_1{+}A_2, B{\times}C}\,{+}\text{R},i \quad\sim\quad \frac{\dfrac{\vdash A_i, B}{\vdash A_1{+}A_2, B}\,{+}\text{R},i \quad \dfrac{\vdash A_i, C}{\vdash A_1{+}A_2, C}\,{+}\text{R},i}{\vdash A_1{+}A_2, B{\times}C}\,\times\text{R}$$

$$\frac{\dfrac{\vdash A, C \quad \vdash A, D}{\vdash A, C{\times}D}\,\times\text{R} \quad \dfrac{\vdash B, C \quad \vdash B, D}{\vdash B, C{\times}D}\,\times\text{R}}{\vdash A{\times}B, C{\times}D}\,\times\text{R} \quad\sim\quad \frac{\dfrac{\vdash A, C \quad \vdash B, C}{\vdash A{\times}B, C}\,\times\text{R} \quad \dfrac{\vdash A, D \quad \vdash B, D}{\vdash A{\times}B, D}\,\times\text{R}}{\vdash A{\times}B, C{\times}D}\,\times\text{R}$$

Figure 2 Cut-free rule permutations.

Since we will be working with a graphical representation, we will adopt **Barendregt's convention**, that bound variable names are globally unique identifiers, in the following form. In a sequent $\vdash A, B$ we assume all quantifiers have a unique binding variable, distinct from any free variable. In a proof π over $\vdash A, B$, a variable x that is universally quantified as $\forall x.C$ in $\vdash A, B$ is an **eigenvariable**. A $\forall$R rule on $\forall x.C$ is considered to bind all free occurrences of the eigenvariable x in its direct subproof. Accordingly, we assume that x does not occur free outside these subproofs (which can be guaranteed by globally renaming x). We take variable names to be persistent throughout a proof, in the sense that we don't admit alpha-conversion (renaming of bound variables) between proof rules. We thus have unique bound variable names in $\vdash A, B$, but in π all $\forall$R rules on the subformula $\forall x.C$ share the same eigenvariable x.

A **link** (C, D) on a sequent $\vdash A, B$ is a pair of subformulas $C \leq A$ and $D \leq B$. A **linking** λ on the sequent $\vdash A, B$ is a set of links on $\vdash A, B$.

▶ **Definition 1.** *A* ***pre-net*** $\lambda \rhd A, B$ *is a sequent* $\vdash A, B$ *with a linking* λ *on it.*

Witness maps

We will record the witnessing terms to existential quantifiers as (explicit) substitutions at each link. A **witness map** $\sigma\colon \textsc{var} \rightharpoonup \textsc{term}$ is a substitution map which assigns terms to variables, given as a (finite) partial function $\sigma = [t_1/x_1, \ldots, t_n/x_n]$. Its **domain** $\textsc{dom}(\sigma)$ is $\{x_1, \ldots, x_n\}$. We abbreviate by $y \in \sigma$ that a variable y occurs free in the range of σ ($y \in \textsc{fv}(t_i)$ for some $i \leq n$). The map $\sigma - x$ is undefined on x and as σ otherwise; $\sigma|_V$ is the restriction of σ to a set of variables V, and $\varnothing$ is the empty witness map. We write $A\sigma$ for the application of the substitutions in σ to the formula A, and $\sigma\tau$ is the **composition** of two maps, where $A(\sigma\tau) = (A\sigma)\tau$. We apply σ to a proof π, written $\pi\sigma$, by applying it to each formula in the proof and to each existential witness t recorded with a rule $\exists$R$, t$.

A **witness linking** λ_Σ is a linking λ with a **witness labelling** $\Sigma\colon \lambda \to \textsc{var} \rightharpoonup \textsc{term}$ that assigns each link (C, D) a witness map. We may use and define λ_Σ as a set of **witness links** $(C, D)_\sigma$ where $(C, D) \in \lambda$ and $\Sigma(C, D) = \sigma$. A witness link $(a, b)_\sigma$ on atomic formulas is an **axiom link** if $\bar{a}\sigma = b\sigma$. An **axiom** witness linking is one comprising axiom links.

▶ **Definition 2.** *A* ***witness pre-net*** $\lambda_\Sigma \rhd A, B$ *is a sequent* $\vdash A, B$ *with a witness linking* λ_Σ*.*

▶ **Definition 3.** *The* ***de-sequentialization*** $[\pi]$ *of a sequent proof* $\pi \vdash A, B$ *is the witness pre-net* $[\pi]^{A,B}_{\varnothing} \rhd A, B$ *where the function* $[-]^{A,B}_{\sigma}$ *is defined inductively in Figure 3 (symmetric cases are omitted).*

A function call $[\pi \vdash A, B]^{A',B'}_{\sigma}$ expects that $A = A'\sigma$ and $B = B'\sigma$: the translation separates a sequent $\vdash A, B$ into subformulas A', B' of the ultimate conclusion of the proof, and the accumulated existential witnesses σ. For an example, we refer to the introduction.

Correctness and sequentialization by coalescence

For sequentialization, the links in a pre-net will be labelled with a sequent proof. An axiom link will carry an axiom, and each coalescence step will introduce one proof rule. Formalizing this, a **proof linking** λ^Π_Σ is a witness linking λ_Σ with a **proof labelling** $\Pi\colon \lambda \to \textsc{proof}$ assigning a sequent proof to each link. We will use and define λ^Π_Σ as a set of **proof links** $(C, D)^\pi_\sigma$, where we require that $\pi \vdash C\sigma, D\sigma$, i.e. that π proves the conclusion $\vdash C\sigma, D\sigma$. A **labelled pre-net** $\lambda^\Pi_\Sigma \rhd A, B$ is a witness pre-net $\lambda_\Sigma \rhd A, B$ with a proof labelling Π on λ_Σ.

$$\left[\frac{}{\vdash a, \overline{a}}\,\text{ax}\right]_{\sigma}^{b\,,\,c} \quad = \quad \{(b,c)_\sigma\}$$

$$\left[\frac{\begin{array}{c}\pi\\ \vdash A, B_i\end{array}}{\vdash A, B_1 + B_2}\,{+}\text{R},i\right]_{\sigma}^{A'\,,\,B_1'+B_2'} \quad = \quad \left[\begin{array}{c}\pi\\ \vdash A, B_i\end{array}\right]_{\sigma}^{A'\,,\,B_i'}$$

$$\left[\frac{\begin{array}{c}\pi\\ \vdash A, B\end{array}\quad\begin{array}{c}\pi'\\ \vdash A, C\end{array}}{\vdash A, B \times C}\,{\times}\text{R}\right]_{\sigma}^{A'\,,\,B'\times C'} \quad = \quad \left[\begin{array}{c}\pi\\ \vdash A, B\end{array}\right]_{\sigma}^{A'\,,\,B'} \cup \left[\begin{array}{c}\pi'\\ \vdash A, C\end{array}\right]_{\sigma}^{A'\,,\,C'}$$

$$\left[\frac{\begin{array}{c}\pi\\ \vdash A, B[t/x]\end{array}}{\vdash A, \exists x.B}\,\exists\text{R},t\right]_{\sigma}^{A'\,,\,\exists x.B'} \quad = \quad \left[\begin{array}{c}\pi\\ \vdash A, B[t/x]\end{array}\right]_{\sigma[t/x]}^{A'\,,\,B'}$$

$$\left[\frac{\begin{array}{c}\pi\\ \vdash A, B\end{array}}{\vdash A, \forall x.B}\,\forall\text{R}\right]_{\sigma}^{A'\,,\,\forall x.B'} \quad = \quad \left[\begin{array}{c}\pi\\ \vdash A, B\end{array}\right]_{\sigma}^{A'\,,\,B'}$$

Figure 3 De-sequentialization.

If λ_Σ is an axiom linking, we assign an **initial proof labelling** $\lambda_\Sigma^\star$ as follows.

$$\lambda_\Sigma^\star = \{\ (a,b)_\sigma^\pi \ \mid\ (a,b)_\sigma \in \lambda_\Sigma\ ,\ \pi = \overline{\vdash a\sigma, b\sigma}\ \}$$

For correctness we may coalesce a pre-net directly, without constructing a proof. So far we have accumulated the following notational conventions:

π, ϕ, ψ	$\in$	PROOF	ALL1 sequent proofs
κ, λ	$\subset$	FORM × FORM	linkings (sets of pairs of formulas)
ρ, σ, τ	:	VAR ⇀ TERM	witness maps
Σ, Θ	:	$\lambda \to$ VAR ⇀ TERM	witness labellings on a linking λ
Π, Φ, Ψ	:	$\lambda \to$ PROOF	proof labellings on a linking λ

▶ **Definition 4.** ***Strict sequentialization*** *($\to$) is the rewrite relation on labelled pre-nets generated by the rules in Figure 4, which replace one or two links by another in a pre-net $\lambda_\Sigma^\Pi \rhd A, B$ where B has a subformula $D_1{+}D_2$, $D_1{\times}D_2$, $\exists x.D$, and $\forall x.D$ respectively. Symmetric cases are omitted.* ***Strict coalescence*** *is the same relation on witness pre-nets, ignoring proof labels, illustrated in Figure 5. A witness pre-net $\lambda_\Sigma \rhd A, B$* ***strict-coalesces*** *if it reduces to $\{(A,B)_\varnothing\} \rhd A, B$. It* ***strongly*** *strict-coalesces if any coalescence path terminates at $\{(A,B)_\varnothing\} \rhd A, B$.*

For an example of coalescence, see the introduction.

▶ **Definition 5.** *An ALL1* ***witness proof net*** *or* ***witness net*** *is a witness pre-net $\lambda_\Sigma \rhd A, B$ with λ_Σ an axiom linking, that strict-coalesces. It* ***sequentializes*** *to a proof π if its initial labelling $\lambda_\Sigma^\star \rhd A, B$ reduces in ($\to$) to $\{(A,B)_\varnothing^\pi\} \rhd A, B$.*

$$(C, D_i)^{\pi}_{\sigma} \to (C, D_1{+}D_2)^{\psi}_{\sigma} \qquad \psi = \dfrac{\begin{matrix}\pi \\ \vdash C\sigma, D_i\sigma\end{matrix}}{\vdash C\sigma, D_1\sigma + D_2\sigma}\,{+}\mathrm{R},i \qquad (+\mathrm{S}, i)$$

$$\left.\begin{matrix}(C, D_1)^{\pi}_{\sigma} \\ (C, D_2)^{\phi}_{\sigma}\end{matrix}\right\} \to (C, D_1{\times}D_2)^{\psi}_{\sigma} \qquad \psi = \dfrac{\begin{matrix}\pi \\ \vdash C\sigma, D_1\sigma\end{matrix} \quad \begin{matrix}\phi \\ \vdash C\sigma, D_2\sigma\end{matrix}}{\vdash C\sigma, D_1\sigma \times D_2\sigma}\,{\times}\mathrm{R} \qquad (\times\mathrm{S})$$

$$(C, D)^{\pi}_{\sigma} \to (C, \exists x.D)^{\psi}_{\sigma - x} \quad (x \in \mathrm{DOM}(\sigma)) \qquad \psi = \dfrac{\begin{matrix}\pi \\ \vdash C\sigma, D\sigma\end{matrix}}{\vdash C(\sigma - x), \exists x.D(\sigma - x)}\,\exists\mathrm{R},\sigma(x) \qquad (\exists\mathrm{S})$$

$$(C, D)^{\pi}_{\sigma} \to (C, \forall x.D)^{\psi}_{\sigma} \quad (x \notin \sigma) \qquad \psi = \dfrac{\begin{matrix}\pi \\ \vdash C\sigma, D\sigma\end{matrix}}{\vdash C\sigma, \forall x.D\sigma}\,\forall\mathrm{R} \qquad (\forall\mathrm{S})$$

Figure 4 Strict coalescence.

Figure 5 Coalescence rules.

We conclude this section by establishing that sequentialization and de-sequentialization for witness nets are inverses, and that witness nets are canonical.

▶ **Theorem 6.** *For any* ALL1 *proof* π*, the witness net* $[\pi]$ *sequentializes to* π*.*

▶ **Theorem 7.** *If* $\lambda_\Sigma \rhd A, B$ *sequentializes to* π*, then* $[\pi]$ *is* $\lambda_\Sigma \rhd A, B$*.*

▶ **Theorem 8.** *Witness nets are canonical:* $[\pi] = [\phi]$ *if and only if* $\pi \sim \phi$*.*

3 Geometric correctness

We will first identify two aspects of sequent proofs, arising from the local nature of the rules, which need to be enforced explicitly in a geometric correctness condition.

Local eigenvariables. The side-condition on the $\forall R$-rule, that the eigenvariable is not free in the context, means that eigenvariables are *local* to the subproof of the $\forall R$-rule. Correspondingly, a link $(C, D)_\sigma$ on $\vdash A, B$ has **local eigenvariables** if for any variable $x \in \sigma$, if x is an eigenvariable quantified as $\forall x.X$ in $\vdash A, B$, then $C \leq X$ or $D \leq X$. A witness linking or pre-net has **local eigenvariables** if all its links do.

Exact coverage. The local witness substitution $[t/x]$ in a rule instance $\exists\mathrm{R}, t$ will have been applied exactly to the axioms $\vdash a, \overline{a}$ in the subproof of that rule. Correspondingly, for a link $(C, D)_\sigma$ on $\vdash A, B$ we expect the domain of σ to be exactly the existential variables in A and B that (could) occur free in C and D. For a subformula C of A, let the **free existential variables** of C in A be the set $\mathrm{EV}_A(C) = \{ x \mid C < \exists x.X \leq A \}$. A link $(C, D)_\sigma$ on $\vdash A, B$ then has **exact coverage** if $\mathrm{DOM}(\sigma) = \mathrm{EV}_A(C) \cup \mathrm{EV}_B(D)$. If it does, σ consists of two components, $\sigma|_{\mathrm{EV}_A(C)}$ and $\sigma|_{\mathrm{EV}_B(D)}$, which we abbreviate as σ_C and σ_D respectively. A witness linking or pre-net has **exact coverage** if all its links do.

Both conditions are captured naturally by coalescence, as can be observed from the rules.

Slices

A **slice** is the fraction of a proof that depends on a given choice of one branch (or projection) on each product formula $A \times B$. Important to additive proof theory is that many operations can be performed on a per-slice basis, such as normalization, or proof net correctness. We will here use slices for the latter purpose. As in the propositional case [21], we define a **slice** of a sequent $\vdash A, B$ as a set of potential links, of which exactly one must be realized in a proof net $\lambda_\Sigma \triangleright A, B$. We extend the propositional criterion in two ways:

Expansion. When defining slices, we interpret an existential quantification $\exists x.A$ as a sum over all witnesses t_i to x that occur in the pre-net, $A[t_1/x] + \ldots + A[t_n/x]$. This captures the non-permutability of a product rule over distinct instantiations, as below left. (Technically, a slice of $\exists x.A$ will correspond to an infinite sum over $A[t/x]$ for every term t, but only the actually occurring terms t_i will ever be relevant.)

Dependency. We define a **dependency** relation between a universal quantification $\forall x.A$ and an instantiation $B[t/y]$ of $\exists y.B$ where the eigenvariable x occurs free in t (a standard approach to first-order quantification [25, 8, 10, 26]). For each link, we will require this dependency relation to be *acyclic*, which amounts to *slice-wise* first-order correctness. It captures the non-permutability of universal and existential sequent rules due to the *eigenvariable condition* of the former, as below right.

$$\dfrac{\dfrac{\vdash A[s/x], B}{\vdash \exists x.A, B}\ \exists\mathrm{R}, s \qquad \dfrac{\vdash A[t/x], C}{\vdash \exists x.A, C}\ \exists\mathrm{R}, t}{\vdash \exists x.A, B \times C}\ \times\mathrm{R} \qquad\qquad \dfrac{\dfrac{\vdash A, B[t/y]}{\vdash A, \exists y.B}\ \exists\mathrm{R}, t}{\vdash \forall x.A, \exists y.B}\ \forall\mathrm{R} \quad \text{where } x \in \mathrm{FV}(t)$$

Expansion covers the interaction between existential quantifiers and products, and *dependency* that between existential and universal quantifiers. Note that in proof nets that include the multiplicatives (e.g. [8, 18]), instead of a dependency as a *partial order* it is common to consider the underlying graph, created by adding a *jump* or *leap* edge when an existential instantiation $B[t/y]$ of $\exists y.B$ contains the eigenvariable x of $\forall x.A$. These edges participate in the *switching* condition [5], to capture the interaction between quantifiers and tensors. Without the tensor, here, the dependency as a partial order suffices.

▶ **Definition 9** (Slice). *A **slice** S of a formula A and a witness map σ is a set of pairs (A', σ'), where $A' \leq A$ and $\sigma' \supseteq \sigma$, given by $S = \{(A, \sigma)\} \cup S'$ where:*

- *If $A = a$ then $S' = \varnothing$.*
- *If $A = B + C$ then $S' = S_B \uplus S_C$ with S_B a slice of B and σ, and S_C one of C and σ.*
- *If $A = B \times C$ then S' is a slice of B and σ or a slice of C and σ.*
- *If $A = \exists x.B$ then $S' = \uplus_{t \in \mathrm{TERM}} S_t$ where each S_t is a slice of B and $\sigma[t/x]$.*
- *If $A = \forall x.B$ then S' is a slice of B and σ.*

*A **slice** of a sequent* $\vdash A, B$ *is a set of links*

$$\{\ (C, D)_{\sigma \cup \tau}\ \mid\ (C, \sigma) \in S_A\ ,\ (D, \tau) \in S_B\ \}$$

where S_A *is a slice of* A *and* $\varnothing$*, and* S_B *a slice of* B *and* $\varnothing$*. A **slice** of a witness pre-net* $\lambda_\Sigma \rhd A, B$ *is the intersection* $\lambda_\Sigma \cap S$ *of* λ_Σ *with a slice* S *of* $\vdash A, B$*.*

As in the propositional case, for correctness we will require that each slice is a singleton. We will further define a **dependency** condition to ensure that the order in which quantifiers are instantiated is sound, corresponding to the *eigenvariable condition* on the $\forall$R-rule of sequent calculus. For simplicity, we define the condition on individual links rather than slices.

▶ **Definition 10.** *In a pre-net* $\lambda_\Sigma \rhd A, B$*, let the **column** of a link* $(C, D)_\sigma$ *be the set of pairs*

$$\{\ (X, \sigma|_{\mathrm{EV}_A(X)})\ \mid\ C \le X \le A\ \} \cup \{\ (Y, \sigma|_{\mathrm{EV}_B(Y)})\ \mid\ D \le Y \le B\ \}\ ,$$

*with a **dependency** relation* $(\preccurlyeq)$*:* $(X, \rho) \preccurlyeq (Y, \tau)$ *if* $X \le Y$ *or* Y *occurs as* $\forall x.Y$ *and* $x \in \rho$*.*

▶ **Definition 11.** *A witness pre-net is **correct** if:*

- *it has local eigenvariables and exact coverage,*
- *it is **slice-correct**: every slice is a singleton, and*
- *it is **dependency-correct**: every column is a partial order (i.e. is acyclic/antisymmetric).*

To conclude this section we will establish that both correctness conditions, by coalescence and by slicing, are equivalent. Moreover, both are equivalent to *strong* coalescence.

▶ **Theorem 12.** *A witness pre-net that strict-coalesces is correct, and a correct witness pre-net strongly strict-coalesces.*

▶ **Corollary 13.** *A correct witness pre-net with axiom linking is a witness proof net.*

4 Composition

We will describe the composition of two witness nets by a global operation. It consists of the relational composition of both linkings, as in the propositional case, where for each pair of links that are being connected, their witness maps are composed. As links correspond to slices, the operation is effectively first-order composition [26] applied slice-wise.

Cut-elimination rules for ALL1 are given in Figure 6; the requisite permutations are in Figure 7.

$\exists v.\overline{P}$ — $[f(x)/y, z/v]$ — $\forall x.\exists y.\forall z.P$
$\exists x.\forall y.\exists z.\overline{P}$ — $[t/x, g(y)/z]$ — P
$\Rightarrow$
$\exists v.\overline{P}$ — $[g(f(t))/v]$ — P

We use the example above to illustrate the composition of links. To eliminate the central cut, on $\forall x.\exists y.\forall z.P$ and $\exists x.\forall y.\exists z.\overline{P}$, the explicit substitutions for both formulas must be effectuated. An inductive procedure, as in sequent calculus, could apply them from outside in: first $[t/x]$, then $[f(t)/y]$ (previously $[f(x)/y]$), then $[g(f(t))/z]$ (previously $[g(y)/z]$).

For a direct definition, to compose two links $(a, b)_\sigma$ and $(\overline{b}, c)_\tau$, the substitutions into the cut-formula σ_b and $\tau_{\overline{b}}$ must be applied as often as needed, up to the depth of quantifiers

$$\dfrac{\dfrac{\begin{matrix}\pi_1\\ \vdash A, B_1\end{matrix} \quad \begin{matrix}\pi_2\\ \vdash A, B_2\end{matrix}}{\vdash A, B_1 \times B_2}{\scriptstyle \times\mathrm{R}} \qquad \dfrac{\begin{matrix}\phi\\ \vdash \overline{B}_i, C\end{matrix}}{\vdash \overline{B}_1 + \overline{B}_2, C}{\scriptstyle +\mathrm{R},i}}{\vdash A, C}{\scriptstyle \mathrm{cut}} \quad \Rightarrow \quad \dfrac{\begin{matrix}\pi_i\\ \vdash A, B_i\end{matrix} \qquad \begin{matrix}\phi\\ \vdash \overline{B}_i, C\end{matrix}}{\vdash A, C}{\scriptstyle \mathrm{cut}}$$

$$\dfrac{\dfrac{\begin{matrix}\pi\\ \vdash A, B[t/x]\end{matrix}}{\vdash A, \exists x.B}{\scriptstyle \exists\mathrm{R},t} \qquad \dfrac{\begin{matrix}\phi\\ \vdash \overline{B}, C\end{matrix}}{\vdash \forall x.\overline{B}, C}{\scriptstyle \forall\mathrm{R}}}{\vdash A, C}{\scriptstyle \mathrm{cut}} \quad \Rightarrow \quad \dfrac{\begin{matrix}\pi\\ \vdash A, B[t/x]\end{matrix} \qquad \begin{matrix}\phi[t/x]\\ \vdash \overline{B}[t/x], C\end{matrix}}{\vdash A, C}{\scriptstyle \mathrm{cut}}$$

Figure 6 ALL1 cut-elimination steps.

$$\dfrac{\dfrac{\vdash A, B}{\vdash \forall x.A, B}{\scriptstyle \forall\mathrm{R}} \quad \vdash \overline{B}, C}{\vdash \forall x.A, C}{\scriptstyle \mathrm{cut}} \;\sim\; \dfrac{\dfrac{\vdash A, B \quad \vdash \overline{B}, C}{\vdash A, C}{\scriptstyle \mathrm{cut}}}{\vdash \forall x.A, C}{\scriptstyle \forall\mathrm{R}}$$

$$\dfrac{\dfrac{\vdash A[t/x], B}{\vdash \exists x.A, B}{\scriptstyle \exists\mathrm{R},t} \quad \vdash \overline{B}, C}{\vdash \exists x.A, C}{\scriptstyle \mathrm{cut}} \;\sim\; \dfrac{\dfrac{\vdash A[t/x], B \quad \vdash \overline{B}, C}{\vdash A[t/x], C}{\scriptstyle \mathrm{cut}}}{\vdash \exists x.A, C}{\scriptstyle \exists\mathrm{R},t}$$

$$\dfrac{\dfrac{\vdash A, B \quad \vdash \overline{B}, C}{\vdash A, C}{\scriptstyle \mathrm{cut}} \quad \vdash \overline{C}, D}{\vdash A, D}{\scriptstyle \mathrm{cut}} \;\sim\; \dfrac{\vdash A, B \quad \dfrac{\vdash \overline{B}, C \quad \vdash \overline{C}, D}{\vdash \overline{B}, D}{\scriptstyle \mathrm{cut}}}{\vdash A, D}{\scriptstyle \mathrm{cut}}$$

$$\dfrac{\dfrac{\vdash A_i, B}{\vdash A_1 + A_2, B}{\scriptstyle +\mathrm{R},i} \quad \vdash \overline{B}, C}{\vdash A_1 + A_2, C}{\scriptstyle \mathrm{cut}} \;\sim\; \dfrac{\dfrac{\vdash A_i, B \quad \vdash \overline{B}, C}{\vdash A_i, C}{\scriptstyle \mathrm{cut}}}{\vdash A_1 + A_2, C}{\scriptstyle +\mathrm{R},i}$$

$$\dfrac{\dfrac{\vdash A_1, B \quad \vdash A_2, B}{\vdash A_1 \times A_2, B}{\scriptstyle \times\mathrm{R}} \quad \vdash \overline{B}, C}{\vdash A_1 \times A_2, C}{\scriptstyle \mathrm{cut}} \;\sim\; \dfrac{\dfrac{\vdash A_1, B \quad \vdash \overline{B}, C}{\vdash A_1, C}{\scriptstyle \mathrm{cut}} \quad \dfrac{\vdash A_2, B \quad \vdash \overline{B}, C}{\vdash A_2, C}{\scriptstyle \mathrm{cut}}}{\vdash A_1 \times A_2, C}{\scriptstyle \times\mathrm{R}}$$

Figure 7 Cut-permutations.

above b, to the terms in the range of the remaining substitutions, σ_a and τ_c. To formalize this, we will use the following notions:

- The **domain-preserving composition** of two witness maps $\sigma \cdot \tau$ is the map $(\sigma\tau)|_{\mathrm{DOM}(\sigma)}$.
- The **least fixed point** $\vec{\sigma}$ of a witness map σ is the least map ρ satisfying $\rho = \rho\sigma$.

The latter is the shortest sequence $\vec{\sigma} = \sigma\sigma \dots \sigma$ such that no variable is both in the domain and range of $\vec{\sigma}$. This is not necessarily finite; in our composition operations, finiteness is ensured by the correctness conditions on proof nets (see Theorem 15).

▶ **Definition 14.** *The* ***composition*** $(A, B)^{\pi}_{\sigma} \,;\, (\overline{B}, C)^{\phi}_{\tau}$ *of two proof links is* $(A, C)^{\psi}_{\rho}$ *where*

$$\rho = \sigma_A \tau_C \cdot \overrightarrow{\sigma_B \tau_{\overline{B}}} \qquad \text{and} \qquad \psi = \dfrac{\left(\begin{matrix}\pi\\ \vdash A\sigma, B\sigma\end{matrix}\right)^{\overrightarrow{\sigma_B \tau_{\overline{B}}}} \quad \left(\begin{matrix}\phi\\ \vdash \overline{B}\tau, C\tau\end{matrix}\right)^{\overrightarrow{\sigma_B \tau_{\overline{B}}}}}{\vdash A\rho, C\rho}{\scriptstyle \mathrm{cut}} \,.$$

The ***composition*** $\lambda^{\Pi}_{\Sigma}\,;\kappa^{\Phi}_{\Theta}$ *of two linkings is the linking*

$$\{\ (X,Y)^{\pi}_{\sigma}\,;(\overline{Y},Z)^{\phi}_{\tau}\ \mid\ (X,Y)^{\pi}_{\sigma}\in\lambda^{\Pi}_{\Sigma}\ ,\ (\overline{Y},Z)^{\phi}_{\tau}\in\kappa^{\Phi}_{\Theta}\ \}$$

The ***composition*** $(\lambda^{\Pi}_{\Sigma}\rhd A,B)\,;(\kappa^{\Phi}_{\Theta}\rhd\overline{B},C)$ *of two pre-nets is the pre-net* $(\lambda^{\Pi}_{\Sigma}\,;\kappa^{\Phi}_{\Theta})\rhd A,C$*. These compositions may omit proof annotations and witness annotations.*

The composition of two links is strongly related to composition of strategies in game semantics. There, two strategies on $\vdash A,B$ and $\vdash\overline{B},C$ are composed by *interaction* on the interface of B and $\overline{B}$, and subsequently *hiding* that interaction.

In the following we will demonstrate that composition gives the desired result: if a net L sequentializes to π and R to ϕ, then $L\,;R$ sequentializes to a normal form of the composition of π and ϕ with a cut. To this end we will explore how composition and sequentialization interact. We will consider the critical pairs of sequentialization ($\to$) with composition ($\Rightarrow$) given in Figures 8–10, and demonstrate how they are resolved.

- $\vdash A,B_1{\times}B_2\ ;\ \vdash\overline{B}_1{+}\overline{B}_2,C$ (Figure 8)
  Since the free existential variables of B and B_1 are the same, $\sigma_B\tau_{\overline{B}}=\sigma_{B_1}\tau_{\overline{B_1}}$ and $\rho=\rho'$. It then follows that ψ' cut-eliminates in one step to ψ.
- $\vdash A,\exists x.B\ ;\ \vdash\forall x.\overline{B},C$ (Figure 9)
  Since x is not free in the range of τ, nor in the range of σ (by Barendregt's convention), we have that $\overrightarrow{\sigma_B\tau_{\overline{B}}}$ is $\overrightarrow{(\sigma_B-x)\tau_{\overline{B}}}$ plus the substitution $[\sigma(x)/x]$. Then $\rho=\rho'$ (as x does not occur in the range of $\sigma_A\tau_C$) and ψ' reduces to ψ in a single cut-elimination step.
- $\vdash A,B\ ;\ \vdash\overline{B},\exists x.C$ (Figure 10)
  Observe that since x occurs in C but not B, it is not in the domain of τ_B, so that τ_B-x is just τ_B. Then $\rho'=\rho-x$, and the diagram is closed by a sequentialization step (from left to right) which extends ψ with an existential introduction rule, to a proof equivalent to ψ':
  $$\dfrac{\begin{matrix}\psi\\ \vdash A\rho,C\rho\end{matrix}}{\vdash A\rho',\exists x.C\rho'}\ \exists\mathrm{R},\rho(x)$$

There are three further critical pairs, for a proof net on $\vdash A,B$ composed with one on $\vdash\overline{B},C_1{+}C_2$, one on $\vdash\overline{B},C_1{\times}C_2$, and one on $\vdash\overline{B},\forall x.C$. These converge like the one above. Resolving these critical pairs gives the soundness of the composition operation, per the following theorem. We abbreviate a cut on proofs $\pi\vdash A,B$ and $\phi\vdash\overline{B},C$ by $\pi\,;\phi$.

▶ **Theorem 15.** *If proof nets* $\lambda_\Sigma\rhd A,B$ *and* $\kappa_\Theta\rhd\overline{B},C$ *sequentialize to* π *and* ϕ *respectively, then their composition* $(\lambda_\Sigma\rhd A,B)\,;(\kappa_\Theta\rhd\overline{B},C)$ *is well-defined (i.e. all fixed points are finite) and sequentializes to a normal form* ψ *of* $\pi\,;\phi$*.*

5 Unification nets

In this final section we explore a second notion of ALL1 proof net: **unification nets** omit any witness information, which is then reconstructed by coalescence. This yields a natural notion of *most general* proof net, where every other proof net is obtained by introducing more witness information. Conversely, every witness net has an underlying unification net, that sequentializes to a *most general* proof.

We consider a proof $\pi\vdash A,B$ **more general** than $\pi'\vdash A,B$, written $\pi\le\pi'$, if there is a substitution map ρ such that $\pi\rho=\pi'$. Unlike for proof nets, this notion is not so natural for

$$A \;[\pi,\sigma \mid \pi',\sigma]\; B_1 \times B_2,\quad \overline{B}_1 + \overline{B}_2 \;[\phi,\tau]\; C \quad\rightarrow\quad A \;[\pi'',\sigma]\; B_1 \times B_2,\quad \overline{B}_1 + \overline{B}_2 \;[\phi',\tau]\; C$$

$$\Downarrow \qquad\qquad \Downarrow$$

$$A \;[\psi,\rho]\; C \qquad\qquad A \;[\psi',\rho']\; C$$

$$\rho = \sigma_A\tau_C \cdot \overrightarrow{\sigma_B\tau_{\overline{B}}}$$

$$\rho' = \sigma_A\tau_C \cdot \overrightarrow{\sigma_{B_1}\tau_{\overline{B_1}}}$$

$$\psi = \dfrac{\left(\dfrac{\pi}{\vdash A\sigma, B_1\sigma}\right)\overrightarrow{\sigma_{B_1}\tau_{\overline{B_1}}} \quad \left(\dfrac{\phi}{\vdash \overline{B}_1\tau, C\tau}\right)\overrightarrow{\sigma_{B_1}\tau_{\overline{B_1}}}}{\vdash A\rho, C\rho}\,\text{cut}$$

$$\psi' = \dfrac{\left(\dfrac{\dfrac{\pi}{\vdash A\sigma, B_1\sigma} \quad \dfrac{\pi'}{\vdash A\sigma, B_2\sigma}}{\vdash A\sigma, B_1\sigma \times B_2\sigma}\right)\overrightarrow{\sigma_B\tau_{\overline{B}}} \quad \left(\dfrac{\dfrac{\phi}{\vdash \overline{B}_1\tau, C\tau}}{\vdash \overline{B}_1\tau + \overline{B}_2\tau, C\tau}\right)\overrightarrow{\sigma_B\tau_{\overline{B}}}}{\vdash A\rho', C\rho'}\,\text{cut}$$

Figure 8 The critical pair $\vdash A, B_1{\times}B_2$; $\vdash \overline{B}_1{+}\overline{B}_2, C$.

$$A \;[\pi,\sigma]\; \exists x.B,\quad \forall x.\overline{B} \;[\phi,\tau]\; C \quad\rightarrow\ (\text{and } \overset{x\notin\tau}{\rightarrow})\quad A \;[\pi',\sigma-x]\; \exists x.B,\quad \forall x.\overline{B} \;[\phi',\tau]\; C$$

$$\Downarrow \qquad\qquad \Downarrow$$

$$A \;[\psi,\rho]\; C \qquad\qquad A \;[\psi',\rho']\; C$$

$$\rho = \sigma_A\tau_C \cdot \overrightarrow{\sigma_B\tau_{\overline{B}}}$$

$$\rho' = \sigma_A\tau_C \cdot \overrightarrow{(\sigma_B - x)\tau_{\overline{B}}}$$

$$\psi = \dfrac{\left(\dfrac{\pi}{\vdash A\sigma, B\sigma}\right)\overrightarrow{\sigma_B\tau_{\overline{B}}} \quad \left(\dfrac{\phi}{\vdash \overline{B}\tau, C\tau}\right)\overrightarrow{\sigma_B\tau_{\overline{B}}}}{\vdash A\rho, C\rho}\,\text{cut}$$

$$\psi' = \dfrac{\left(\dfrac{\dfrac{\pi}{\vdash A\sigma, B\sigma}}{\vdash A\sigma, \exists x.B\sigma - x}\right)\overrightarrow{(\sigma_B - x)\tau_{\overline{B}}} \quad \left(\dfrac{\dfrac{\phi}{\vdash \overline{B}\tau, C\tau}}{\vdash \forall x.\overline{B}\tau, C\tau}\right)\overrightarrow{(\sigma_B - x)\tau_{\overline{B}}}}{\vdash A\rho', C\rho'}\,\text{cut}$$

Figure 9 The critical pair $\vdash A, \exists x.B$; $\vdash \forall x.\overline{B}, C$.

$$A \;[\pi,\sigma]\; B,\quad \overline{B} \;[\phi,\tau]\; \exists x.C \quad\rightarrow\quad A \;[\pi,\sigma]\; B,\quad \overline{B} \;[\phi',\tau-x]\; \exists x.C$$

$$\Downarrow \qquad\qquad \Downarrow$$

$$A \;[\psi,\rho]\; \exists x.C \qquad\qquad A \;[\psi',\rho']\; \exists x.C$$

$$\rho = \sigma_A\tau_C \cdot \overrightarrow{\sigma_B\tau_{\overline{B}}}$$

$$\rho' = \sigma_A(\tau_C - x) \cdot \overrightarrow{\sigma_B\tau_{\overline{B}}}$$

$$\psi = \dfrac{\left(\dfrac{\pi}{\vdash A\sigma, B\sigma}\right)\overrightarrow{\sigma_B\tau_{\overline{B}}} \quad \left(\dfrac{\phi}{\vdash \overline{B}\tau, C\tau}\right)\overrightarrow{\sigma_B\tau_{\overline{B}}}}{\vdash A\rho, C\rho}\,\text{cut}$$

$$\psi' = \dfrac{\left(\dfrac{\pi}{\vdash A\sigma, B\sigma}\right)\overrightarrow{\sigma_B\tau_{\overline{B}}} \quad \left(\dfrac{\dfrac{\phi}{\vdash \overline{B}\tau, C\tau}}{\vdash \overline{B}\tau, \exists x.C(\tau - x)}\right)\overrightarrow{\sigma_B\tau_{\overline{B}}}}{\vdash A\rho', \exists x.C\rho'}\,\text{cut}$$

Figure 10 The critical pair $\vdash A, B$; $\vdash \overline{B}, \exists x.C$.

sequent proofs: in the permutation of existential and product rules below, from left to right u must be generated as the least term more general than s and t; from right to left, s and t cannot be reconstructed from u, and must be retrieved from their respective subproofs.

$$\dfrac{\dfrac{\vdash A, C}{\vdash A, \exists x.C}\,\exists\mathrm{R},s \qquad \dfrac{\vdash B, C}{\vdash B, \exists x.C}\,\exists\mathrm{R},t}{\vdash A \times B, \exists x.C}\,\times\mathrm{R} \quad \sim \quad \dfrac{\dfrac{\vdash A, C \quad \vdash B, C}{\vdash A \times B, C}\,\times\mathrm{R}}{\vdash A \times B, \exists x.C}\,\exists\mathrm{R},u$$

To reconstruct witnesses by unification, we define the following operations.

$\sigma \leq \tau$: A witness map σ is **more general** than τ if there is a map ρ such that $\sigma\rho = \tau$.

$\sigma \frown \tau$: Two witness maps σ and τ are **coherent** if there is a map ρ such that $\sigma\rho = \tau\rho$.

$\sigma \vee \tau$: The **join** of coherent witness maps is the least map ρ such that $\sigma \leq \rho$ and $\tau \leq \rho$.

A link (a, b) on two atomic formulas is an **axiom** link if there exists a witness map σ such that $\overline{a}\sigma = b\sigma$. To an axiom link (a, b) over $\vdash A, B$ we assign an **initial witness map**, which is the least witness map σ over the domain $\mathrm{EV}_A(a) \cup \mathrm{EV}_B(b)$ such that $\overline{a}\sigma = b\sigma$. In other words, σ is the most general unifier of $\overline{a}$ and b, over the given domain, written $\mathrm{MGU}(\overline{a}, b)$. For an axiom linking λ over $\vdash A, B$ the **initial witness pre-net** $\lambda_\star \rhd A, B$ is given by

$$\lambda_\star = \{ \ (a, b)_\sigma \ \mid \ (a, b) \in \lambda \ , \ \sigma = \mathrm{MGU}(\overline{a}, b) \ \} \ .$$

An initial witness pre-net has *exact coverage*, while coalescence will give *local eigenvariables*. Note that eigenvariables are constants for the purpose of unification (they are not substituted into). For Barendregt's convention, that free variables have distinct names from bound ones, we should assume that variables in the range of a witness map are fresh; for example, the most general unifier of existential variables x and y should be $[z/x, z/y]$ for a fresh variable z, and not $[y/x]$ or $[x/y]$.

▶ **Definition 16.** ***Unifying sequentialization*** *($\rightsquigarrow$) is the rewrite relation on labelled pre-nets generated by the rules* $(+\mathrm{U}, i)$, $(\exists\mathrm{U})$, $(\forall\mathrm{U})$, *which are respectively as* $(+\mathrm{S}, i)$, $(\exists\mathrm{S})$, *and* $(\forall\mathrm{S})$, *and the rule*

$$\left.\begin{array}{l}(C, D_1)^\pi_\sigma \\ (C, D_2)^\phi_\tau\end{array}\right\} \rightsquigarrow (C, D_1 {\times} D_2)^\psi_{\sigma\vee\tau = \sigma\rho = \tau\rho} \qquad (\sigma\frown\tau) \qquad \psi = \dfrac{\left(\overset{\pi}{\vdash A\sigma, B\sigma}\right)\rho \quad \left(\overset{\phi}{\vdash A\tau, C\tau}\right)\rho}{\vdash A(\sigma\vee\tau), B\sigma\rho \times C\tau\rho} \qquad (\times\mathrm{U})$$

Unifying coalescence *is the relation ($\rightsquigarrow$) on witness pre-nets, ignoring proof labels. A witness pre-net $\lambda_\Sigma \rhd A, B$* ***unifying-coalesces*** *if it reduces to $\{(A, B)_\varnothing\} \rhd A, B$ and* ***strongly*** *unifying-coalesces if any coalescence path terminates at $\{(A, B)_\varnothing\} \rhd A, B$.*

▶ **Definition 17.** *An* ALL1 ***unification proof net*** *or* ***unification net*** *is a pre-net $\lambda \rhd A, B$ with axiom linking λ such that the initial witness pre-net $\lambda_\star \rhd A, B$ unifying-coalesces. It* ***sequentializes*** *to π if $\lambda^\star_\star \rhd A, B$ reduces in ($\rightsquigarrow$) to $\{(A, B)^\pi_\varnothing\} \rhd A, B$.*

In the above definition, note that $\lambda^\star_\star = (\lambda_\star)^\star$ is the initial proof labelling of $\lambda_\star$, which assigns an axiom rule to each axiom link. For a minimal example, see the Introduction. Observe also that unifying coalescence includes strict coalescence, $(\rightarrow) \subseteq (\rightsquigarrow)$. The following two lemmata relate sequentialization for witness nets and unification nets.

▶ **Lemma 18.** *In ($\rightsquigarrow$), if $\lambda_\Sigma \rhd A, B$ sequentializes to π then $\lambda_\star \rhd A, B$ sequentializes to $\pi' \leq \pi$.*

▶ **Lemma 19.** *If $\lambda_\star \rhd A, B$ unifying-sequentializes to π then there exists a witness assignment Σ and substitution ρ such that $\lambda_\Sigma \rhd A, B$ strict-sequentializes to π and $\lambda_\Sigma = \lambda_\star\rho$.*

	Monomial nets	Witness nets	Unification nets
Canonicity	?	✓	✓
Generality	✗	✗	✓
Direct composition	✗	✓	✓
Coalescence	?	✓	✓
Slicing	✓	✓	?

Figure 11 Comparison of different notions of proof nets for quantifiers and additive connectives.

We can then show that sequentialization and de-sequentialization for unification nets are inverses up to generality, and that composition is sound.

▶ **Theorem 20.** *If* $[\pi \vdash A, B]$ *is* $\lambda_\Sigma \triangleright A, B$ *then* $\lambda \triangleright A, B$ *unifying-sequentializes to* $\pi' \leq \pi$*.*

▶ **Theorem 21.** *If* $\lambda \triangleright A, B$ *sequentializes to* π*, then* $[\pi] = \lambda_\Sigma \triangleright A, B$ *for some* Σ*.*

▶ **Theorem 22.** *If* $\lambda \triangleright A, B$ *sequentializes to* π *and* $\kappa \triangleright \overline{B}, C$ *to* ϕ *then their composition* $\lambda\, ; \kappa \triangleright A, C$ *sequentializes to a proof* $\psi' \leq \psi$ *where* ψ *is a normal form of* $\pi\, ; \phi$*.*

6 Conclusion and related work

We have presented two notions of first-order additive proof net, *witness nets* and *unification nets*, to capture canonically two natural notions of proof identity for first-order additive linear logic. Figure 11 summarizes our results, along with some observations we make below.

Proof nets with additives and quantifiers existed before as *monomial nets* [8]. These are not generally canonical: they admit the permutation (and duplication) of proof rules past implicit *contractions* (that is, the shared context of the additive conjunction rule). However, it might be possible to restrict additive monomial nets to some notion of canonical form. Likewise, coalescence (or contractibility) has not been studied for first-order monomial nets, though it has been extended to a related form of MALL proof nets [24].

Our slicing condition is loosely related to a number of approaches to first-order *classical* logic. A formula $\exists x.A$ is interpreted as the sum (or classically, the *disjunction*) over a fixed number of instantiations $A[t_1/x] + \ldots + A[t_n/x]$. This can be traced to Herbrand's Theorem [14]: $\exists x.A$ is equivalent to the infinite sum over $A[t/x]$ for all terms t in the language, but for any given proof a finite set of terms suffices. *Expansion tree proofs* [25, 10] are a graphical proof formalism based on this idea. In our slicing condition, the interpretation of $\exists x.A$ is an infinite sum of which only a finite part $A[t_1/x] + \ldots + A[t_n/x]$ is relevant, over the witnesses $t_1, \ldots, t_n$ actually assigned to x in the proof net. An interesting alternative approach to additive proof nets, which we may explore in future work, is to take the expansion of $\exists x.A$ to $A[t_1/x] + \ldots + A[t_n/x]$ as primary, and record it explicitly in the syntax, as expansion tree proofs do for classical logic. It is expected, however, that this would sacrifice *generality* and *direct composition*.

Hetzl [15] explores *explicit substitution* for first-order classical sequent calculus.

Lambek observed that the two canonical proofs of $\vdash A{+}A, \overline{A}$ can be distinguished by casting each as a specialization (by substituting into propositional variables) of the more general proofs of $\vdash a{+}b, \overline{a}$ and $\vdash a{+}b, \overline{b}$ (corresponding to the two injections of a sum). He proposed to use this idea of *generality* as the basis for a notion of *proof identity*: two proofs are equivalent if their most general forms are isomorphic [23]. How this extends to first order is not obvious. The natural first-order analogue of $\vdash A{+}A, \overline{A}$ would be $\vdash \exists x.A, \overline{A}$ where the quantifier is vacuous, as $\exists x.A$ represents the infinite sum over A (for all terms t). Where

Lambek's generality *distinguishes* the two proofs of $\vdash A{+}A, \overline{A}$, ours *identifies* the proofs of $\vdash \exists x.A, \overline{A}$: the sequent has one unification net, but infinitely many witness nets (one for each term t). If existential quantification is indeed analogous to a sum, Lambek's notion of generality is more faithfully captured by witness nets than unification nets.

For future work, there are a few natural questions. First is that of a geometric criterion for unification nets. For this, it seems essential to reconstruct dependencies between products and existential quantifiers globally, as the *expansion* condition provides for witness nets. These interact in highly intricate ways, which the unifying coalescence algorithm resolves incrementally and locally. To do so globally, as would be necessary for a geometric criterion, is a major combinatorial challenge.

A second is whether coalescence (or *contractibility* [4]) applies to MLL1 unification nets [18]. We believe it would, straightforwardly (without requiring a dependency or leap edges).

A final question is whether the current approach can be extended to obtain proof nets for MALL1. We see two ways forward that could succeed: (i) combine witness nets with MALL slice nets [21] using a slicing criterion; (ii) combine witness nets or unification nets with MALL conflict nets [20] using a coalescence criterion. For both, there are still significant combinatorial challenges, for example because coalescence for MALL is not a straightforward extension of that for ALL. Other approaches are much less certain: MALL conflict nets do not yet have a slicing criterion (though this looks feasible), and MALL slice nets do not support coalescence (which seems fundamentally problematic).

References

1 Samson Abramsky. Sequentiality vs. Concurrency in Games and Logic. *Mathematical Structures in Computer Science*, 13(4):531–565, 2003.

2 Luís Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In *CONCUR*, volume 6269 of *LNCS*, pages 222–236, 2010.

3 Robin Cockett and Luigi Santocanale. On the Word Problem for ΣΠ-Categories, and the Properties of Two-Way Communication. In *Computer Science Logic (CSL), 18th Annual Conference of the EACSL*, pages 194–208, 2009.

4 Vincent Danos. *La Logique Linéaire appliquée à l'étude de divers processus de normalisation (principalement du Lambda-calcul)*. PhD thesis, Université Paris 7, 1990.

5 Vincent Danos and Laurent Regnier. The structure of multiplicatives. *Archive for Mathematical Logic*, 28:181–203, 1989.

6 Didier Galmiche and Jean-Yves Marion. Semantic Proof Search Methods for ALL – a first approach –. Short paper in Theorem Proving with Analytic Tableaux, 4th International Workshop (TABLEAUX'95). Available from the first author's webpage, 1995.

7 Jean-Yves Girard. Linear Logic. *Theoretical Computer Science*, 50(1):1–102, 1987.

8 Jean-Yves Girard. Proof-nets: the parallel syntax for proof-theory. *Logic and Algebra*, pages 97–124, 1996.

9 Stefano Guerrini and Andrea Masini. Parsing MELL proof nets. *Theoretical Computer Science*, 254(1-2):317–335, 2001.

10 Willem Heijltjes. Classical proof forestry. *Ann. Pure Appl. Logic*, 161(11):1346–1366, 2010.

11 Willem Heijltjes. Proof nets for additive linear logic with units. In *26th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 207–216, 2011.

12 Willem Heijltjes and Dominic J. D. Hughes. Complexity bounds for sum–product logic via additive proof nets and Petri nets. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 80–91, 2015.

13 Willem Heijltjes, Dominic J. D. Hughes, and Lutz Straßburger. Proof nets for first-order additive linear logic. Technical Report RR-9201, INRIA, 2018. URL: `hal.inria.fr/hal-01867625/`.

14 Jacques Herbrand. Investigations in proof theory: The properties of true propositions. In Jean van Heijenoort, editor, *From Frege to Gödel: A source book in mathematical logic, 1879–1931*, pages 525–581. Harvard University Press, 1967.
15 Stefan Hetzl. A sequent calculus with implicit term representation. In *Computer Science Logic (CSL)*, volume 6247 of *LNCS*, pages 351–365, 2010.
16 Kohei Honda, Vasco Vasconcelos, and Makoto Kubo. Language primitives and type disciplines for structured communication-based programming. In *European Symposium on Programming*, pages 122–138, 1998.
17 Dominic J. D. Hughes. Proofs Without Syntax. *Annals of Mathematics*, 164(3):1065–1076, 2006.
18 Dominic J. D. Hughes. Unification nets: canonical proof net quantifiers. In *33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 2018.
19 Dominic J. D. Hughes. First-order proofs without syntax. Available at arXiv.org, 2019.
20 Dominic J. D. Hughes and Willem Heijltjes. Conflict nets: efficient locally canonical MALL proof nets. In *31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 2016.
21 Dominic J. D. Hughes and Rob van Glabbeek. Proof nets for unit-free multiplicative-additive linear logic. *Transactions on Computational Logic*, 6(4):784–842, 2005.
22 Andre Joyal. Free Lattices, Communication and Money Games. *Proc. 10th Int. Cong. of Logic, Methodology and Philosophy of Science*, 1995.
23 Joachim Lambek. Deductive Systems and Categories I, II, III. *Theory of Computing Systems (I), Lecture Notes in Mathematics (II, III)*, 1968–1972.
24 Roberto Maieli. Retractile Proof Nets of the Purely Multiplicative and Additive Fragment of Linear Logic. In *14th Int. Conf. Logic for Programming Artificial Intelligence and Reasoning (LPAR)*, pages 363–377, 2007.
25 Dale Miller. A Compact Representation of Proofs. *Studia Logica*, 46(4):347–370, 1987.
26 Samuel Mimram. The Structure of First-Order Causality. *Mathematical Structures in Computer Science*, 21(1):65–110, 2011.
27 Lutz Straßburger. A Characterisation of Medial as Rewriting Rule. In Franz Baader, editor, *Term Rewriting and Applications, RTA'07*, volume 4533 of *LNCS*, pages 344–358, 2007.
28 Philip Wadler. Propositions as Sessions. *Journal of Functional Programming*, 24(2-3):384–418, 2014.

A Proofs

In this appendix we restate and prove all theorems, and add two supporting lemmata.

▶ **Theorem 6.** *For any* ALL1 *proof* π*, the witness net* $[\pi]$ *sequentializes to* π*.*

Proof. It follows by induction on π that if $\lambda_\Sigma = [\pi \vdash A, B]^{A',B'}_\sigma$ where $A'\sigma = A$ and $B'\sigma = B$, then $\lambda^\star_\Sigma \triangleright A, B$ reduces in $(\to)$ to $\{(A', B')^\pi_\sigma\} \triangleright A, B$. The statement is the case $\sigma = \varnothing$. ◀

▶ **Theorem 7.** *If* $\lambda_\Sigma \triangleright A, B$ *sequentializes to* π*, then* $[\pi]$ *is* $\lambda_\Sigma \triangleright A, B$*.*

Proof. By induction on the sequentialization path $\lambda^\star_\Sigma \triangleright A, B \to^* \{(A, B)^\pi_\varnothing\} \triangleright A, B$ it follows that in every pre-net $\kappa^\Phi_\Theta \triangleright A, B$ on this path, λ_Σ is equal to the union over the de-sequentialization of all proof labels ϕ in Φ:

$$\lambda_\Sigma = \bigcup \{ [\phi]^{C,D}_\sigma \mid (C, D)^\phi_\sigma \in \kappa^\Phi_\Theta \} .$$

The statement is then the case $\kappa^\Phi_\Theta = \{(A, B)^\pi_\varnothing\}$. ◀

▶ **Theorem 8.** *Witness nets are canonical:* $[\pi] = [\phi]$ *if and only if* $\pi \sim \phi$*.*

Proof. From left to right is by inspection of the critical pairs of sequentialization ($\rightarrow$). From right to left is by inspection of the rule permutations in Figure 2. ◀

▶ **Lemma 23.** *Strict coalescence preserves and reflects correctness.*

Proof. For a strict coalescence step $L \rightarrow R$, we will show that the witness pre-net L is correct if and only if R is. Let $L = \lambda_\Sigma \rhd A, B$ and $R = \kappa_\Theta \rhd A, B$. In each case, exact coverage and local eigenvariables are immediately preserved and reflected. For slice-correctness, we will demonstrate that the left-hand side and right-hand side of each rule belong to the same slice of $\vdash A, B$, or in the case of $\exists$S, naturally corresponding slices. For dependency-correctness, we will briefly show how acyclicity of the columns of the involved links is preserved.

- $(C, D_i)_\sigma \rightarrow (C, D_1{+}D_2)_\sigma$
  A slice S_B of B and $\varnothing$ containing one of (D_1, τ), (D_2, τ), and $(D_1 + D_2, \tau)$ must also contain the other two. A slice S of $\vdash A, B$ then contains all three of $(C, D_1)_\sigma$, $(C, D_2)_\sigma$, and $(C, D_1{+}D_2)_\sigma$, or none. It follows that $S \cap \lambda_\Sigma$ is a singleton if and only if $S \cap \kappa_\Theta$ is. Since other slices are unaffected, L is slice-correct if and only if R is.
  For dependency-correctness, the column of $(C, D_i)_\sigma$ is that of $(C, D_1{+}D_2)_\sigma$ plus the pair $(D_i, \sigma|_{\mathrm{EV}_B(D_i)})$ itself, which is minimal in the order $\preccurlyeq$.
- $(C, D_1)_\sigma, (C, D_2)_\sigma \rightarrow (C, D_1{\times}D_2)_\sigma$
  A slice S of $\vdash A, B$ contains $(C, D_1{\times}D_2)_\sigma$ if and only if it contains either of $(C, D_1)_\sigma$ or $(C, D_2)_\sigma$, and cannot contain both. Then $S \cap \lambda_\Sigma$ is a singleton if and only if $S \cap \kappa_\Theta$ is. Dependency-correctness is immediate, as above.
- $(C, D)_\sigma \rightarrow (C, \exists x.D)_{\sigma - x}$
  A slice S of $\vdash A, B$ contains $(C, \exists x.D)_\tau$ if and only if it contains all links $(C, D)_{\tau[t/x]}$ for any term t. Letting $\tau[t/x] = \sigma$, then $S \cap \lambda_\Sigma$ is the singleton $\{(C, D)_\sigma\}$ if and only if $S \cap \kappa_\Theta$ is $\{(C, \exists x.D)_{\sigma - x}\}$.
  For dependency-correctness, the column of $(C, D)_\sigma$ is that of $(C, \exists x.D)_{\sigma - x}$ plus a pair (D, τ), which is minimal in $(\preccurlyeq)$.
- $(C, D)_\sigma \rightarrow (C, \forall x.D)_\sigma$
  A slice S of $\vdash A, B$ contains $(C, D)_\sigma$ if and only if it contains also $(C, \forall x.D)_\sigma$, and hence $S \cap \lambda_\Sigma$ is a singleton if and only if $S \cap \kappa_\Theta$ is.
  For dependency-correctness, the column of $(C, D)_\sigma$ is that of $(C, \forall x.D)_\sigma$ plus a pair (D, τ). The side-condition of the coalescence step is that $x \notin \sigma$; then x does not occur free in any (X, ρ), and (D, τ) is minimal in $(\preccurlyeq)$. ◀

▶ **Lemma 24.** *To a correct witness pre-net $\lambda_\Sigma \rhd A, B$ a coalescence step applies, unless it is fully coalesced already, $\lambda_\Sigma = \{(A, B)_\varnothing\}$.*

Proof. Let the **depth** of a link $(C, D)_\sigma$ be a pair of integers (n, m), where n is the distance from C to the root of A, and m that from D to B. We order link depth in the product order: $(i, j) \leq (n, m)$ if and only if $i \leq n$ and $j \leq m$. We will demonstrate that a link at maximal depth may always be coalesced, unless it is the unique link $(A, B)_\varnothing$ at $(0, 0)$.

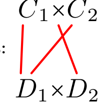
To see that a maximally deep link coalesces, first note that a link $(C, D_i)_\sigma$ where D_i occurs in $D_0{+}D_1$ may always coalesce, as may a link $(C, D)_\sigma$ where D occurs in $\exists x.D$. This leaves the following cases:

- $(A, D_i)_\sigma$ with D_i occurring in $D = D_1 \times D_2$.
  Without loss of generality, let $i = 1$. A slice S_1 of $\vdash A, B$ containing $(A, D_1)_\sigma$ has a counterpart S_2 containing $(A, D_2)_\sigma$. The depth of $(A, D_2)_\sigma$ is the same as that of $(A, D_1)_\sigma$. By correctness $S_2 \cap \lambda_\Sigma$ is a singleton; by the assumption of maximality it

may not contain a deeper link than $(A,D_2)_\sigma$; and it may not contain a shallower one since that would be shared with $S_1 \cap \lambda_\Sigma$. Then $\lambda_\Sigma \triangleright A,B$ contains both $(A,D_1)_\sigma$ and $(A,D_2)_\sigma$, and these contract to $(A,D)_\sigma$.

- $(A,D)_\sigma$ with D in $\forall x.D$.
  The step $(A,D)_\sigma \to (A,\forall x.D)_\sigma$ applies if $x \notin \sigma$. By way of contradiction, assume $x \in \sigma$. The column of $(A,D)_\sigma$ contains (D,σ_D) and $(\forall x.D,\tau)$ where $\tau = \sigma|_{\text{EV}_B(\forall x.D)}$. By the exact coverage condition, $\sigma = \sigma_A \cup \sigma_D$, and since the free existential variables in D and $\forall x.D$ are the same, $\text{EV}_B(D) = \text{EV}_B(\forall x.D)$, so that $\tau = \sigma_D$. (Note that since $\sigma_A = \varnothing$, we get $\sigma = \sigma_D = \tau$, but this is not essential to the argument.) Since $x \in \sigma$ we have $x \in \tau$, and in the column of $(A,D)_\sigma$ we have $(\forall x.D,\tau) \preccurlyeq (D,\tau)$ since D occurs as $\forall x.D$. But we already have $(D,\tau) \preccurlyeq (\forall x.D,\tau)$ because $D \leq \forall x.D$, contradicting antisymmetry of $(\preccurlyeq)$. Then $x \notin \sigma$, and the step $(A,D)_\sigma \to (A,\forall x.D)_\sigma$ applies.
- $(C_i,D_j)_\sigma$ in $C = C_1 \times C_2$ and $D = D_1 \times D_2$.
  Without loss of generality, let $i = j = 1$. By minimal depth and using similar reasoning to the first case above, the pre-net must contain one of the following three configurations.
  1. $(C_1,D_1)_\sigma,(C_1,D_2)_\sigma,(C_2,D_1)_\sigma,(C_2,D_2)_\sigma$
  2. $(C_1,D_1)_\sigma,(C_1,D_2)_\sigma,(C_2,D)_\sigma$
  3. $(C_1,D_1)_\sigma,(C_2,D_1)_\sigma,(C,D_2)_\sigma$

  1: $C_1{\times}C_2$ / $D_1{\times}D_2$

  3: $C_1{\times}C_2$ / $D_1{\times}D_2$

  In the second case, the step $(C_1,D_1)_\sigma,(C_1,D_2)_\sigma \to (C_1,D)_\sigma$ applies; in the third case, $(C_1,D_1)_\sigma,(C_2,D_1)_\sigma \to (C,D_1)_\sigma$; and in the first case, both.
- $(C_i,D)_\sigma$ in $C = C_1 \times C_2$ and $\forall x.D$.
  Without loss of generality let $i = 1$. If $x \notin \sigma$ the rewrite step $(C_1,D)_\sigma \to (C_1,\forall x.D)_\sigma$ applies. Otherwise, let $x \in \sigma$. The slice S_1 of $\vdash A,B$ containing $(C_1,D)_\sigma$ has a counterpart S_2 containing $(C_2,D)_\sigma$, which must include exactly one link of λ_Σ. By the assumption of minimal depth, it cannot have greater depth than $(C_2,D)_\sigma$. It cannot be $(C,D)_\sigma$ or any shallower link, since that would be shared with the slice S_1 which already contains $(C_1,D)_\sigma$. It cannot be $(C_2,\forall x.D)_\sigma$ or any shallower link $(C_2,X)_\tau$ (i.e. with $\forall x.D \leq X$) because $x \in \sigma$. This would mean either $x \in \tau$ which contradicts the *eigenvariables not free* convention, or $x \in \text{FV}(\sigma(y))$ where $\forall x.D < \exists y.Y \leq X$ which creates a cyclic column, as in the second case above. It follows that $S_2 \cap \lambda_\Sigma = \{(C_2,D)_\sigma\}$, so that the rewrite step $(C_1,D)_\sigma,(C_2,D)_\sigma \to (C,D)_\sigma$ applies.
- $(C,D)_\sigma$ in $\forall x.C$ and $\forall y.D$.
  A rewrite step $(C,D)_\sigma \to (\forall x.C,D)_\sigma$ or $(C,D)_\sigma \to (C,\forall y.D)_\sigma$ applies unless $x,y \in \sigma$. But that would generate a cycle in the column of $(C,D)_\sigma$, in one of three ways. If $x \in \sigma_C$ or $y \in \sigma_D$ then, since $\sigma_C = \sigma_{\forall x.C}$ and $\sigma_D = \sigma_{\forall y.D}$, respectively:

  $$(C,\sigma_C) \preccurlyeq (\forall x.C,\sigma_C) \preccurlyeq (C,\sigma_C) \qquad (D,\sigma_D) \preccurlyeq (\forall y.D,\sigma_D) \preccurlyeq (D,\sigma_D)\ .$$

  Otherwise, if $x \in \sigma_D$ and $y \in \sigma_C$ then

  $$(C,\sigma_C) \preccurlyeq (\forall x.C,\sigma_C) \preccurlyeq (D,\sigma_D) \preccurlyeq (\forall x.D,\sigma_D) \preccurlyeq (C,\sigma_C)\ .$$ ◀

▶ **Theorem 12.** *A witness pre-net that strict-coalesces is correct, and a correct witness pre-net strongly strict-coalesces.*

Proof. For the first statement, we proceed by induction on the coalescence path from $\lambda_\Sigma \triangleright A,B$ to $\{(A,B)_\varnothing\} \triangleright A,B$, with the end result as the base case. It is slice-correct: every slice of $\vdash A,B$ contains $(A,B)_\varnothing$, so every slice of $\{(A,B)_\varnothing\} \triangleright A,B$ is the singleton $\{(A,B)_\varnothing\}$. It is also dependency-correct: the column of $(A,B)_\varnothing$ is the set $\{(A,\varnothing),(B,\varnothing)\}$, where A and B are unrelated in $(\preccurlyeq)$. For the inductive step, by Lemma 23 coalescence

reflects correctness, so that any pre-net along the coalescence path is correct, in particular $\lambda_\Sigma \triangleright A, B$.

For the second statement, let $\lambda_\Sigma \triangleright A, B$ be correct. By Lemma 24 either the net has coalesced, or a coalescence step applies. By Lemma 23 the result of any coalescence step is again correct. Since links strictly move towards the roots of both formula trees, it follows that this process terminates, and the pre-net $\lambda_\Sigma \triangleright A, B$ strongly strict-coalesces. ◀

▶ **Theorem 15.** *If proof nets $\lambda_\Sigma \triangleright A, B$ and $\kappa_\Theta \triangleright \overline{B}, C$ sequentialize to π and ϕ respectively, then their composition $(\lambda_\Sigma \triangleright A, B) ; (\kappa_\Theta \triangleright \overline{B}, C)$ is well-defined (i.e. all fixed points are finite) and sequentializes to a normal form ψ of $\pi ; \phi$.*

Proof. By Theorem 12 the proof nets $L = \lambda_\Sigma \triangleright A, B$ and $R = \kappa_\Theta \triangleright \overline{B}, C$ strongly coalesce. We may then interleave their coalescence sequences as follows: if a synchronized step in L and R on the interface B and $\overline{B}$ is available, apply it; otherwise perform steps in L on A and in R on C until it is. This gives the following combined sequence.

$$\begin{array}{ccccccccc} L & = & L_1 & \to^? & L_2 & \to^? & \dots & \to^? & L_n \\ R & = & R_1 & \to^? & R_2 & \to^? & \dots & \to^? & R_n \\ \Downarrow & & \Downarrow & & \Downarrow & & & & \Downarrow \\ L ; R & = & L_1 ; R_1 & \to & L_2 ; R_2 & \to & \dots & \to & L_n ; R_n \end{array}$$

(Here, $(\to^?)$ is the relation $(\to) \cup (=)$, but we assume that at least $L_i \to L_{i+1}$ or $R_i \to R_{i+1}$.) The path along the top and right of this diagram sequentializes L to π' and R to ϕ' (equivalent to π and ϕ respectively), and then composes to $L_n ; R_n = \{(A, C)^{\pi' ; \phi'}_{\varnothing}\} \triangleright A, C$.

Each square of the diagram converges as one of the critical pairs of sequentialization and composition discussed above. Then each path along the diagram from top left (L and R) to bottom right ($L_n ; R_n$) gives a sequentialization, with cuts, of $L_n ; R_n$. Let the path taking the vertical step from L_i and R_i to $L_i ; R_i$ sequentialize to ψ_i, so that $\psi_n = \psi'$. By the way each square converges, we have that ψ_i is reached from ψ_{i+1} by a cut-elimination or permutation step.

Finally, in L and R every link is an axiom link. Any link in $L ; R$ is composed from two links $(a, b)_\sigma$ in L and $(\overline{b}, c)_\tau$ in R, which yields $(a, c)_\rho$ where $\rho = \sigma_a \tau_c \cdot \overrightarrow{\sigma_b \tau_{\overline{b}}}$. This sequentializes to the axiom $\overline{\vdash a\rho, c\rho}$, which is in normal form. Then $L ; R$ is a proof net (it has an axiom linking and it coalesces), and it sequentializes to a normal form of ψ. ◀

▶ **Lemma 18.** *In $(\rightsquigarrow)$, if $\lambda_\Sigma \triangleright A, B$ sequentializes to π then $\lambda_\star \triangleright A, B$ sequentializes to $\pi' \leq \pi$.*

Proof. The sequentialization path $\lambda^\star_\Sigma \triangleright A, B = L_1 \rightsquigarrow L_2 \rightsquigarrow \dots \rightsquigarrow L_n = (A, B)^\pi_\varnothing \triangleright A, B$ has a corresponding path $\lambda^\star_\star \triangleright A, B = R_1 \rightsquigarrow R_2 \rightsquigarrow \dots \rightsquigarrow R_n = (A, B)^{\pi'}_\varnothing \triangleright A, B$ where the same links (but with potentially different witness maps) are coalesced. It follows by induction on this path (where the base case is L_1 and R_1) that for every corresponding pair of links $(C, D)^\phi_\sigma$ in L_i and $(C, D)^\psi_\tau$ in R_i we have $\tau \leq \sigma$ and $\psi \leq \phi$. ◀

▶ **Lemma 19.** *If $\lambda_\star \triangleright A, B$ unifying-sequentializes to π then there exists a witness assignment Σ and substitution ρ such that $\lambda_\Sigma \triangleright A, B$ strict-sequentializes to π and $\lambda_\Sigma = \lambda_\star \rho$.*

Proof. By induction on the sequentialization path $\lambda_\star \triangleright A, B \rightsquigarrow^* (A, B)^\pi_\varnothing \triangleright A, B$. For the end result, the statement holds with $\rho = \varnothing$. For the inductive step, consider a step $L \rightsquigarrow R$. We show the case $(\times\mathrm{U})$; the other cases are immediate.

- $(C, D_1)_\sigma, (C, D_2)_\tau \rightsquigarrow (C, D_1 \times D_2)_{\sigma \vee \tau}$
  By the inductive hypothesis, $R\rho'$ strict-sequentializes to π. Let $\sigma \vee \tau = \sigma\rho'' = \tau\rho''$ and let $\rho = \rho''\rho'$. Then $L\rho$ strict-sequentializes to π by

$$(C, D_1)_{\sigma\rho}, (C, D_2)_{\tau\rho} \to (C, D_1 \times D_2)_{(\sigma \vee \tau)\rho'} \; .$$ ◀

▶ **Theorem 20.** *If $[\pi \vdash A, B]$ is $\lambda_\Sigma \rhd A, B$ then $\lambda \rhd A, B$ unifying-sequentializes to $\pi' \leq \pi$.*

Proof. By Theorem 6, $\lambda_\Sigma \rhd A, B$ sequentializes to π in $(\to)$, and hence also in $(\rightsquigarrow)$. Then by Lemma 18 $\lambda_\star \rhd A, B$ sequentializes to $\pi' \leq \pi$. ◀

▶ **Theorem 21.** *If $\lambda \rhd A, B$ sequentializes to π, then $[\pi] = \lambda_\Sigma \rhd A, B$ for some Σ.*

Proof. By Lemma 19, since $\lambda \rhd A, B$ sequentializes to π there is a net $\lambda_\Sigma \rhd A, B$ that sequentializes to π. By Theorem 7, $[\pi] = \lambda_\Sigma \rhd A, B$. ◀

▶ **Theorem 22.** *If $\lambda \rhd A, B$ sequentializes to π and $\kappa \rhd \overline{B}, C$ to ϕ then their composition $\lambda \,;\, \kappa \rhd A, C$ sequentializes to a proof $\psi' \leq \psi$ where ψ is a normal form of $\pi \,;\, \phi$.*

Proof. By Lemma 19 there are witness labellings Σ and Θ such that $\lambda_\Sigma \rhd A, B$ strict-sequentializes to π and $\kappa_\Theta \rhd \overline{B}, C$ to ϕ. By Theorem 15 their composition $(\lambda_\Sigma \,;\, \kappa_\Theta) \rhd A, C$ strict-sequentializes to a normal form ψ of $\pi \,;\, \phi$. By Lemma 18 the net $(\lambda \,;\, \kappa)_\star \rhd A, C$ unifying-sequentializes to $\psi' \leq \psi$. ◀

The Sub-Additives: A Proof Theory for Probabilistic Choice extending Linear Logic

Ross Horne
Computer Science and Communications, University of Luxembourg, Esch-sur-Alzette, Luxembourg
ross.horne@uni.lu

Abstract

Probabilistic choice, where each branch of a choice is weighted according to a probability distribution, is an established approach for modelling processes, quantifying uncertainty in the environment and other sources of randomness. This paper uncovers new insight showing probabilistic choice has a purely logical interpretation as an operator in an extension of linear logic. By forbidding projection and injection, we reveal additive operators between the standard *with* and *plus* operators of linear logic. We call these operators the *sub-additives*. The attention of the reader is drawn to two sub-additive operators: the first being sound with respect to probabilistic choice; while the second arises due to the fact that probabilistic choice cannot be self-dual, hence has a de Morgan dual counterpart. The proof theoretic justification for the sub-additives is a cut elimination result, employing a technique called *decomposition*. The justification from the perspective of modelling probabilistic concurrent processes is that implication is sound with respect to established notions of probabilistic refinement, and is fully compositional.

2012 ACM Subject Classification Theory of computation → Proof theory; Theory of computation → Process calculi; Theory of computation → Linear logic

Keywords and phrases calculus of structures, probabilistic choice, probabilistic refinement

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.23

Acknowledgements I thank Bogdan Aman and Gabriel Ciobanu for their enjoyable discussions.

1 Introduction

This paper lays down a novel foundation for a proof theory of formulae modelling concurrent processes with mixed probabilistic and non-deterministic choice. Probabilistic choices refine non-deterministic choices by indicating the probability with which one action or another occurs, and have been introduced in game theory and process calculi to model measurable uncertainly in the environment, such as a decision made by tossing a coin.

It is already well known that, in various *processes-as-formulae* approaches to modelling processes using extensions of linear logic [15], the additive operators can be used to model non-deterministic choices. The key novelty of this work is the observation that probabilistic choices can also be handled using additive operators, of a more restrictive kind, which we call the *sub-additives*.

In what follows we clarify the processes-as-formulae approach to modelling processes directly as formulae in extensions of linear logic. We highlight key observations leading to probabilistic sub-additive operators, and explain why their proof theory is non-trivial. Furthermore, for readers for whom the discovery of a novel proof theory is insufficient motivation, we highlight that, unlike most semantics previously proposed for probabilistic concurrent processes, our model is exceptionally compositional, admitting *action refinement*.

© Ross Horne;
licensed under Creative Commons License CC-BY
4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019).
Editor: Herman Geuvers; Article No. 23; pp. 23:1–23:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1.1 The processes-as-formulae paradigm

Various approaches to modelling processes by directly embedding them as formulae in an extension of linear logic have been floated since the discovery of linear logic (see [22] for a comparison). Progress in this processes-as-formulae approach has been accelerated by an advance in proof theory – the *calculus of structures* [17] – a generalisation of the sequent calculus. Process models not limited to CCS [3], session types [5], attack trees [21] and the π-calculus [23, 24] have been tackled using the processes-as-formulae approach.

An advantage of the processes-as-formulae paradigm is that formulae modelling processes can be directly compared using *implication* in the logical system. Furthermore, there are *no design decisions*, since the semantics are determined by the principles of *cut elimination*. In every process model this approach always leads us to a preorder over processes with appealing properties. The preorder obtained enjoys the following properties: it is a congruence; is sound with respect to most commonly-used process preorders, including weak simulation [22], and pomset ideals [21]; and respects *action refinement* – the ability to refine atomic actions with larger sub-processes. This makes implication highly *compositional*.

In this work, by introducing an operator modelling probabilistic choice, the above properties can also be achieved in the probabilistic setting, where preorders are defined with respect to probability distributions. To emphasise this point we prove that implication in this work is sound with respect to a notion of refinement called weak *probabilistic simulation* [37, 2]. A famous result in the theory of probabilistic processes [10], means that, equivalently, implication is sound with respect to *probabilistic may testing* [27, 30]. An advantage implication has over simulation/testing semantics is that, as mentioned above, implication guarantees a greater degree of compositionality.

1.2 Motivation: uncovering the probabilistic sub-additive operators

We explain key observations that uncover the probabilistic sub-additive operators. Sub-additive operators are restricted forms of additive conjunction or disjunction, found in linear logic. Sub-additives forbid projection and injection, while permitting other properties of the additives, notably idempotency.

Firstly, consider how the standard additives can be used to model non-deterministic choice. To be specific, in linear logic, we have *with* $\&$, which enjoys the following *projection laws*, where $\multimap$ is linear implication: $P \& Q \multimap P$ and $P \& Q \multimap Q$. For example, $\mathit{heads} \& \mathit{tails}$ can be used to model a process that does not toss a coin but instead chooses on which side to lay the coin. This can be refined by process *heads* that always chooses to lay down heads. This does not model tossing a coin, instead modelling a decision the process can make.

The key observation is, by restricting additives such that **projection and injection are forbidden**, we are able to model probabilistic choice. For example, $\mathit{heads} \oplus_{1/2} \mathit{tails}$ models a fair coin, where heads or tails occurs with probability 1/2. Notice the process cannot influence the outcome of the coin toss, therefore such a fair coin cannot be refined to *heads*. The absence of this refinement corresponds to forbidding projection. Furthermore, it is standard for probabilistic processes, that a fair coin **cannot** be refined to an unfair coin where the balance of probabilities are different from 1/2 each. Notions of probabilistic refinement preserve the balance of probabilities.

Although projection/injection are forbidden, non-deterministic choice and probabilistic choice are related. For example, non-deterministic choice $\mathit{heads} \& \mathit{tails}$ can be refined to probabilistic choice $\mathit{heads} \oplus_{1/2} \mathit{tails}$. This refinement can be established by proving the following using the logical system in the body of this work.

$$\mathit{heads} \& \mathit{tails} \multimap \mathit{heads} \oplus_{1/2} \mathit{tails}$$

Such a refinement, introducing probabilities, is standard for *probabilistic simulation* or, equivalently, *probabilistic may testing* [27, 30, 9].

Note, there are many other modelling capabilities of the logic in this work. For example, we can capture probabilistic choice with margins of error, and probabilistic model checking, within a bound of probability. Application wise, such models have been used for a wide range of problems, e.g., quantifying the degree of anonymity offered by privacy protocols, or quantifying risk in attacker models. This work focusses on introducing our new logical system ΔMAV and providing clear and simple examples.

The interplay between the sub-additives and both sequential and parallel composition can be non-trivial. For example, we discover, for subtle reasons explained later, in the presence of parallel composition, operator $\oplus_p$ cannot be self-dual. Thereby we obtain also a de Morgan dual operator $\&_p$, essential for completing the symmetry demanded by a logic satisfying cut elimination. The central result of this paper, cut elimination (Theorem 2), ensures these new sub-additive operators co-exist happily with other operators of linear logic – a prerequisite for using implication with confidence. Furthermore, the soundness of linear implication as a notion of probabilistic refinement (Theorem 4) is verified and the merits of this notion of refinement discussed. In particular, we claim that this logical approach to modelling processes helps us discover the coarsest notion of refinement, in the literature, that can: firstly, handle probabilistic processes; secondly, accommodate parallel composition; and, thirdly, permit *action refinement* [41].

Outline of the paper. Section 2, provides established background material on probabilistic processes. Section 3, recalls MALL in the calculus of structures, and introduces the extended system ΔMAV featuring a pair of sub-additive operators. Section 4 provides a series of examples illustrating how we can construct, more traditional, probabilistic simulations from proofs in ΔMAV. Section 5, outlines the proof of cut elimination, necessary to justify the logical system proposed. Section 6 highlights the existence of further sub-additive operators between the standard operators of linear logic.

2 Background: an established notion of probabilistic simulation

We begin with background on probabilistic simulation. We select a minimal probabilistic process calculus and standard notion of probabilistic simulation.

Note there are numerous probabilistic calculi in the literature mixing non-deterministic and probabilistic choice, not limited to probabilistic extensions of CCS [28], CSP [11], and the π-calculus [33]. Due to the rich proof calculi developed [23], expressive process models can be handled by techniques in this work. For scientific clarity, we select here a minimal calculus in order to make a clear comparison with the new logical approach to probabilistic refinement introduced in subsequent sections.

The syntax of our minimal process calculus is drawn from terms in the following grammar, where 'a' represents actions.

$$\begin{aligned} t \;::=\;& \texttt{ok}\ (\text{successful completion}) \mid a.t\ (\text{action prefix}) \mid t \parallel t\ (\text{parallel composition}) \\ \mid\;& t \sqcap t\ (\text{non-deterministic choice}) \mid t +_p t\ (\text{probabilistic choice}) \end{aligned}$$

Discrete probability distributions are uniquely determined by a *probability mass function* $\Delta : S \to [0,1]$ over a set S of process terms such that $\sum_{t \in S} \Delta(t) = 1$. A Dirac distribution for process term s, written $\mathbf{1}_s$, is defined by the probability mass function such that $\Delta(s) = 1$. For probability p and distributions, Δ_1 and Δ_2 *linear combination* $p\Delta_1 + (1-p)\Delta_2$, defined as $(p\Delta_1 + (1-p)\Delta_2)(t) = p\Delta_1(t) + (1-p)\Delta_2(t)$, is a distribution and *dot product* $\Delta_1 \cdot \Delta_2$ is defined such that $(\Delta_1 \cdot \Delta_2)(t \parallel u) = \Delta_1(t)\Delta_2(u)$ and 0 elsewhere.

Process terms are mapped to distributions using the following function δ.

$$\delta(\mathtt{ok}) = \mathbf{1}_{\mathtt{ok}} \qquad \delta(a.t) = \mathbf{1}_{a.t} \qquad \delta(t \sqcap t) = \mathbf{1}_{t \sqcap t}$$
$$\delta(t +_p t) = p\delta(t) + (1-p)\delta(t) \qquad \delta(t \parallel t) = \delta(t) \cdot \delta(t)$$

Labelled transitions from process terms to distributions are defined by the following rules, where label α ranges over any action a or τ.

$$\frac{}{a.t \xrightarrow{a} \delta(t)} \qquad \frac{i \in \{1,2\}}{t_1 \sqcap t_2 \xrightarrow{\tau} \delta(t_i)} \qquad \frac{t_1 \xrightarrow{\alpha} \Delta}{t_1 \parallel t_2 \xrightarrow{\alpha} \Delta \cdot \delta(t_2)} \qquad \frac{t_2 \xrightarrow{\alpha} \Delta}{t_1 \parallel t_2 \xrightarrow{\alpha} \delta(t_1) \cdot \Delta}$$

Labelled transitions lift to weak transitions over distributions, as according to the following four clauses, which allow zero or more τ-transitions. Firstly, $\Delta \xRightarrow{\tau} \Delta$; secondly, if for all i, $s_i \xrightarrow{\alpha} \Delta_i$ and $\sum_{i \in I} p_i = 1$ then $\sum_{i \in I} p_i \mathbf{1}_{t_i} \xRightarrow{\alpha} \sum_i p_i \Delta_i$; thirdly, if $\Delta_1 \xRightarrow{\tau} \Delta_2$ then $p\Delta_1 + (1-p)\mathcal{E} \xRightarrow{\tau} p\Delta_2 + (1-p)\mathcal{E}$, fourthly, if $\Delta_1 \xRightarrow{\tau} \Delta_2$ and $\Delta_2 \xRightarrow{\alpha} \Delta_3$, then $\Delta_1 \xRightarrow{\alpha} \Delta_3$.

For tighter results, we also employ the predicate $\checkmark$ indicating successful *termination*, defined such that $\mathtt{ok}\checkmark$ and if $t_1\checkmark$ and $t_2\checkmark$ then $(t_1 \parallel t_2)\checkmark$. Termination extends to distributions in the obvious way such that if $t\checkmark$ then $\mathbf{1}_t\checkmark$ and if $\Delta\checkmark$ and $\mathcal{E}\checkmark$ then $(p\Delta + (1-p)\mathcal{E})\checkmark$.

The above labelled transitions and termination predicate are employed in the following definition of a *weak complete probabilistic simulation*. The definition also employs a standard *lifting* of relations from processes to distributions.

▶ **Definition 1.** *For a relation $\mathcal{R}$ between processes and distributions, its lifting $\hat{\mathcal{R}}$ is such that: if, for all i, $t_i \mathrel{\mathcal{R}} \Delta_i$ and $\sum_{i \in I} p_i = 1$, then $\sum_{i \in I} p_i \mathbf{1}_{t_i} \mathrel{\hat{\mathcal{R}}} \sum_{i \in I} p_i \Delta_i$. A relation between processes and distributions $\mathcal{R}$ is a weak complete probabilistic simulation whenever:*

- *If $s \mathrel{\mathcal{R}} \Delta$ and $s \xrightarrow{\alpha} \mathcal{E}$, there exists $\mathcal{E}'$ such that $\Delta \xRightarrow{\alpha} \mathcal{E}'$ and $\mathcal{E} \mathrel{\hat{\mathcal{R}}} \mathcal{E}'$.*
- *If $t \mathrel{\mathcal{R}} \Delta$ and $t\checkmark$ then there exists $\mathcal{E}$ such that $\Delta \xRightarrow{\tau} \mathcal{E}$ and $\mathcal{E}\checkmark$.*

If there exists weak complete probabilistic simulation $\mathcal{R}$ such that $\delta(t_1) \mathrel{\hat{\mathcal{R}}} \delta(t_2)$, then we say t_2 simulates t_1.

We refer to the above notion simply as *probabilistic simulation* throughout this work. Recall this definition is used only as a reference to show the logic we develop is sound with respect to such a standard notion of probabilistic refinement, and contains no new concepts. We provide examples later in subsequent sections when making such a comparison.

3 Extending linear logic with probabilistic sub-additive operators

In this section, we introduce a proof system featuring the probabilistic sub-additives. The system is a conservative extension of multiplicative-additive linear logic (MALL). Therefore, first we recall a presentation of MALL in the calculus of structures, a generalisation of the sequent calculus. We employ the calculus of structures, since it provides additional expressive power demanded by our target logic ΔMAV.

3.1 An established presentation of MALL in the calculus of structures

The fragment of linear logic MALL was one of the first proof systems studied in the calculus of structures [38]. Fig 1 recalls a proof system for multiplicative-additive linear logic MALL in the calculus of structures. Inference rules apply in any context. We assume formulae are always in negation-normal-form, where negation is always pushed to atoms, a, by the following function, inducing De Morgan dualities.

$$\overline{P \oplus Q} = \overline{P} \mathbin{\&} \overline{Q} \qquad \overline{P \mathbin{\&} Q} = \overline{P} \oplus \overline{Q} \qquad \overline{\overline{a}} = a \qquad \overline{P \otimes Q} = \overline{P} \mathbin{⅋} \overline{Q} \qquad \overline{P \mathbin{⅋} Q} = \overline{P} \otimes \overline{Q} \qquad \overline{\circ} = \circ$$

The formulation of MALL in Fig. 1 was employed to prove cut elimination for a non-commutative extension of MALL called MAV [20]. The rules are also similar to a version used to study focussing in the calculus of structures [4].

structural congruence:

$$P \parr Q \equiv Q \parr P \qquad (P \parr Q) \parr R \equiv P \parr (Q \parr R) \qquad \circ \parr P \equiv P$$

$$P \otimes Q \equiv Q \otimes P \qquad (P \otimes Q) \otimes R \equiv P \otimes (Q \otimes R) \qquad \circ \otimes P \equiv P$$

inference rules:

$$\frac{\mathcal{C}\{\ \circ\ \}}{\mathcal{C}\{\ \overline{a} \parr a\ \}}\ \text{interact} \qquad \frac{\mathcal{C}\{\ (P \parr Q) \otimes R\ \}}{\mathcal{C}\{\ P \parr (Q \otimes R)\ \}}\ \text{switch} \qquad \frac{\mathcal{C}\{\ \circ\ \}}{\mathcal{C}\{\ \circ \,\&\, \circ\ \}}\ \text{tidy}$$

$$\frac{\mathcal{C}\{\ P_1\ \}}{\mathcal{C}\{\ P_1 \oplus P_2\ \}}\ \text{choose left} \qquad \frac{\mathcal{C}\{\ P_2\ \}}{\mathcal{C}\{\ P_1 \oplus P_2\ \}}\ \text{choose right} \qquad \frac{\mathcal{C}\{\ (P \parr R) \,\&\, (Q \parr R)\ \}}{\mathcal{C}\{\ (P \,\&\, Q) \parr R\ \}}\ \text{external}$$

Figure 1 Structural congruence and inference rules for MALL in the calculus of structures.

The structural congruence ensures the multiplicatives *par* $\parr$ and *times* $\otimes$ are commutative monoids with a common unit. The *switch* rule and *interact* rule form multiplicative linear logic. Regarding the inference rules, there is one rule, *choose*, for additive *plus* $\oplus$, which chooses either the left or right branch during proof search. The rule *external* distributes the additive *with* $\&$ over *par*, forcing both branches to be explored. The *tidy* rule ensures proof search is successful only if both branches are successful.

A *derivation* is a sequence of zero or more rule instances, where the structural congruence can be applied at any step. The bottommost formula is the *conclusion* and the topmost is the *premiss*. A proposition P is *provable*, written $\vdash P$, whenever there exists a derivation with conclusion P and premise $\circ$. *Linear implication* $P \multimap Q$ is defined as $\overline{P} \parr Q$; hence a provable linear implication is written $\vdash P \multimap Q$.

This presentation of MALL has a common unit for the multiplicatives, consequently implication $\vdash P \otimes Q \multimap P \parr Q$ holds. The reader familiar with linear logic will observe this means the *mix* rule is admissible. Note the results in this paper also hold for a formulation of MALL that does not admit *mix*, but *mix* is included so as the logic extends immediately to non-commutative logic.

3.2 Extending with the probabilistic sub-additives (and sequentiality)

The calculus of structures provides a setting in which the sub-additives can be expressed and evaluated. We explain briefly the new rules of the structural congruence and the inference rules in Fig. 2. Note we assume a probability p is always such that $0 < p < 1$, thus any sub-formula that appears in a probabilistic choice occurs with non-zero probability.

The rule of the structural congruence for the probabilistic sub-additives, Fig. 2, ensures the balance of probabilities is maintained when applying idempotency, associativity and commutativity. By maintaining the balance of probabilities, structural congruence preserves underlying probability distributions. For example $p\Delta + (1-p)\Delta = \Delta$, hence we have a weighted form of idempotency $P \oplus_p P = P$.

For associativity, observe if Δ_0, Δ_1 and Δ_2 are distributions corresponding to P, Q and R respectively, then $q\,(p\Delta_0 + (1-p)\Delta_1) + (1-q)\Delta_2 = r\Delta_0 + (1-r)\,(s\Delta_1 + (1-s)\Delta_2)$ only if $r = pq$ and $(1-r)s = q(1-p)$. Furthermore, commuting formulae inverts probabilities $(p\Delta_1 + (1-p)\Delta_2 = (1-p)\Delta_2 + p\Delta_1)$.

structural congruence:

$$P \mathbin{\&_r} Q \equiv Q \mathbin{\&_{1-r}} P \qquad P \mathbin{\&_r} P \equiv P \qquad (P \mathbin{\&_p} Q) \mathbin{\&_q} R \equiv P \mathbin{\&_{pq}} \left(Q \mathbin{\&_{\frac{q(1-p)}{1-pq}}} R\right)$$

$$P \oplus_r Q \equiv Q \oplus_{1-r} P \qquad P \oplus_r P \equiv P \qquad (P \oplus_p Q) \oplus_q R \equiv P \oplus_{pq} \left(Q \oplus_{\frac{q(1-p)}{1-pq}} R\right)$$

$$\circ \triangleleft P \equiv P \qquad P \equiv P \triangleleft \circ \qquad (P \triangleleft Q) \triangleleft R \equiv P \triangleleft (Q \triangleleft R)$$

inference rules:

$$\frac{\mathcal{C}\{\ (P \parr R) \mathbin{\&_p} (Q \parr S)\ \}}{\mathcal{C}\{\ (P \oplus_p Q) \parr (R \mathbin{\&_p} S)\ \}}\ \text{confine}$$

$$\frac{\mathcal{C}\{\ (P \parr R) \oplus_q (Q \parr S)\ \}}{\mathcal{C}\{\ (P \oplus_q Q) \parr (R \oplus_q S)\ \}}\ \text{medial} \qquad \frac{\mathcal{C}\{\ (P \mathbin{\&_p} R) \oplus_q (Q \mathbin{\&_p} S)\ \}}{\mathcal{C}\{\ (P \oplus_q Q) \mathbin{\&_p} (R \oplus_q S)\ \}}\ \text{medial}$$

$$\frac{\mathcal{C}\{\ (P \mathbin{\&} R) \oplus_q (Q \mathbin{\&} S)\ \}}{\mathcal{C}\{\ (P \oplus_q Q) \mathbin{\&} (R \oplus_q S)\ \}}\ \text{medial} \qquad \frac{\mathcal{C}\{\ (P \mathbin{\&} R) \mathbin{\&_p} (Q \mathbin{\&} S)\ \}}{\mathcal{C}\{\ (P \mathbin{\&_p} Q) \mathbin{\&} (R \mathbin{\&_p} S)\ \}}\ \text{medial}$$

$$\frac{\mathcal{C}\{\ (P \parr R) \triangleleft (Q \parr S)\ \}}{\mathcal{C}\{\ (P \triangleleft Q) \parr (R \triangleleft S)\ \}}\ \text{medial} \qquad \frac{\mathcal{C}\{\ (P \mathbin{\&} R) \triangleleft (Q \mathbin{\&} S)\ \}}{\mathcal{C}\{\ (P \triangleleft Q) \mathbin{\&} (R \triangleleft S)\ \}}\ \text{medial}$$

$$\frac{\mathcal{C}\{\ (P \mathbin{\&_p} R) \triangleleft (Q \mathbin{\&_p} S)\ \}}{\mathcal{C}\{\ (P \triangleleft Q) \mathbin{\&_p} (R \triangleleft S)\ \}}\ \text{medial} \qquad \frac{\mathcal{C}\{\ (P \triangleleft R) \oplus_p (Q \triangleleft S)\ \}}{\mathcal{C}\{\ (P \oplus_p Q) \triangleleft (R \oplus_p S)\ \}}\ \text{medial}$$

linear negation:

$$\overline{P \triangleleft Q} = \overline{P} \triangleleft \overline{Q} \qquad \overline{P \oplus_p Q} = \overline{P} \mathbin{\&_p} \overline{Q} \qquad \overline{P \mathbin{\&_p} Q} = \overline{P} \oplus_p \overline{Q}$$

Figure 2 Rules for the probabilistic sub-additive operators and *seq* in ΔMAV, extending Fig. 1.

A self-dual non-commutative operator *seq*, notated $\triangleleft$, is introduced in order to model processes with action prefixes or sequential composition. Seq was first introduced in system BV [17], which was subsequently extended with the additives to obtain system MAV [20]. The operator seq lies between multiplicative operators *times* $\otimes$ and *par* $\parr$ from linear logic [15].

Inference rule *confine* and the *medial* rules are best explained in the context of examples throughout the remainder of this paper. Notice all medials have a standard form.

$$\frac{(P \sqcap R) \sqcup (Q \sqcap S)}{(P \sqcup Q) \sqcap (R \sqcup S)}\ \text{medial} \qquad \text{where } (\sqcap, \sqcup) \in \{\ (\parr, \oplus_q), (\&_p, \oplus_q), (\&, \oplus_q), (\&, \&_p), (\parr, \triangleleft), (\&, \triangleleft), (\&_p, \triangleleft), (\triangleleft, \oplus_q)\ \}$$

Cut elimination in the calculus of structures is equivalent to the following statement.

▶ **Theorem 2** (cut elimination). *In ΔMAV, if* $\vdash \mathcal{C}\{\ P \otimes \overline{P}\ \}$, *then* $\vdash \mathcal{C}\{\ \circ\ \}$.

The above theorem is the main technical justification for the correctness of ΔMAV. A proof sketch is delayed until Section 5. As with MALL, linear implication $P \multimap Q$ is defined in terms of negation and *par* such that $\overline{P} \parr Q$. A useful but straightforward property is linear implication is reflexive. Amongst the immediate consequences of cut elimination is linear

implication in ΔMAV is transitive. Furthermore, also as a corollary of cut elimination, linear implication holds in every context (note negation and implication are derived operators, hence are not part of the syntax of contexts).

▶ **Corollary 3.** *Linear implication is a preorder that holds in every context (a precongruence).*

This corollary establishes a key criteria for using linear implication as a notion of refinement.

Note, in this paper, operator $\&_p$ is treated as a synthetic dual to $\oplus_p$ necessary for completing the proof system, and used when proving linear implications. This operator likely has applications, for modelling probabilistic communicating systems; but we avoid controversy by sticking to the indisputable established probabilistic choice modelled by $\oplus_p$.

3.3 Embedding of Probabilistic Processes in ΔMAV

While cut elimination proves we have made the correct choices of rules for the logic to work, it says little about its relationship to probabilistic refinement. Here we state the main result showing that implication is sound with respect to the key established notions of refinement for probabilistic processes.

We employ the following embedding, mapping processes to formulae.[1]

Name of operator	Process term	Logical operator
success	$[\![\mathtt{ok}]\!]$	$\circ$
prefix	$[\![\alpha.t]\!]$	$\alpha \triangleleft [\![t]\!]$
parallel composition	$[\![t_1 \parallel t_2]\!]$	$[\![t_1]\!] \otimes [\![t_2]\!]$
external choice	$[\![t_1 \sqcap t_2]\!]$	$[\![t_1]\!] \& [\![t_2]\!]$
probabilistic choice	$[\![t_1 +_p t_2]\!]$	$[\![t_1]\!] \oplus_p [\![t_2]\!]$

The mapping extends to discrete probability distributions over process terms such that $[\![\mathbf{1}_t]\!] = [\![t]\!]$ and if $\Delta = p\Delta_1 + (1-p)\Delta_2$, where $0 < p < 1$ then $[\![\Delta]\!] = [\![\Delta_1]\!] \oplus_p [\![\Delta_2]\!]$.

Using the above embedding of processes as formulae we can compare processes using linear implication. All linear implications between processes can also be established using weak complete probabilistic simulation. Each approach is quite different, since the former involves unfolding logical rules while the latter involves defining a simulation relation witnessing the refinement. Here these two approaches to probabilistic refinement are formally connected as follows.

▶ **Theorem 4.** *If* $\vdash [\![t_1]\!] \multimap [\![t_2]\!]$*, in* Δ*MAV, then* t_1 *simulates* t_2 *(Def. 1).*

The proof provides a proceedure that constructs a weak complete probabilistic simulation from any linear implications between embeddings of processes. It adapts proof techniques devised for establishing a similar results for the π-calculus [22] (without probabilities).

The converse of Theorem 4 does not hold. As reinforced by related work [21], linear implication has non-interleaving properties. For example $a \parr a \multimap a \triangleleft a$ does **not** hold, but these processes are equivalent in any interleaving semantics, including probabilistic simulation in Def. 1. This can be regarded as a strength of linear implication, since such non-interleaving semantics are preserved under *action refinement* [41] – the substitution of an atomic action with any process. For the minimal process language in this this work, we consider only refinement of an action with a sequence of actions.

[1] Note the system is completely symmetric so the dual operators could be used, inverting implication.

▶ **Corollary 5.** *For process terms t_1 and t_2, and substitution σ mapping actions, say a, to a sequence of actions, say $b_1 \cdot\cdots\cdot b_n$, if $\vdash [\![t_1]\!] \multimap [\![t_2]\!]$ then $\vdash [\![t_1\sigma]\!] \multimap [\![t_2\sigma]\!]$.*

For example, since $\vdash [\![a \parallel a]\!] \multimap [\![a.a]\!]$ holds, by applying the action refinement $\sigma = \left\{{}^{b.c}/_{a}\right\}$, the following holds: $\vdash [\![b.c \parallel b.c]\!] \multimap [\![b.c.b.c]\!]$.

Action refinement is not respected by any interleaving semantics, including weak complete probabilistic simulation (previous work on action refinement in the probabilistic setting [8] avoids parallel composition). Furthermore, although there is work on probabilistic event structures [1, 42], linear implication in ΔMAV appears to be the first non-interleaving notion of refinement accommodating probabilistic choice.

4 Examples of properties established using linear implication

Having introduced definitions and stated the main results, we illustrate the theory with examples. This section covers examples of refinements that are permitted or forbidden between processes. There are also some examples justifying the medial rules.

4.1 Refinements also provable using probabilistic simulation

As noted in the introduction, projection and injection are forbidden for probabilistic simulation, hence should be forbidden for the sub-additives. Indeed, the following processes are unrelated by linear implication.

$$\textit{heads} +_{1/2} \textit{tails} \quad \text{is unrelated to} \quad \textit{heads} \quad \text{and also is unrelated to} \quad \textit{tails}$$

Hence, as a consequence of Theorem 4, **none** of the following hold in general: $P \multimap P \oplus_p P$, $P \oplus_p Q \multimap P$, $Q \multimap P \oplus_p P$ and $P \oplus_p Q \multimap Q$.

Now, using the rules of ΔMAV, we can verify the following chain of implications, proving that the probabilistic sub-additives lie between the standard additives.

$$P \mathbin{\&} Q \multimap P \mathbin{\&_p} Q \qquad\qquad P \mathbin{\&_p} Q \multimap P \oplus_p Q \qquad\qquad P \oplus_p Q \multimap P \oplus Q$$

The first implication has a proof of the following form.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\circ}{\circ \mathbin{\&_p} \circ}\ \textit{idempotency}}{\left(\overline{P} \parr P\right) \mathbin{\&_p} \left(\overline{Q} \parr Q\right)}\ \text{Proposition 3}}{\left(\left(\overline{P} \oplus \overline{Q}\right) \parr P\right) \mathbin{\&_p} \left(\left(\overline{P} \oplus \overline{Q}\right) \parr Q\right)}\ \textit{choose}}{\left(\left(\overline{P} \oplus \overline{Q}\right) \oplus_p \left(\overline{P} \oplus \overline{Q}\right)\right) \parr \left(P \mathbin{\&_p} Q\right)}\ \textit{confine}}{\left(\overline{P} \oplus \overline{Q}\right) \parr \left(P \mathbin{\&_p} Q\right)}\ \textit{idempotency}$$

Also, due to de Morgan dualities, the third implication in the chain above has a proof of the same form (by setting P as $\overline{P}$ and Q as $\overline{Q}$). The second implication in the chain of implications above has the following proof.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\circ}{\circ \mathbin{\&_p} \circ}\ \textit{idempotency}}{\left(\overline{P} \parr P\right) \mathbin{\&_p} \left(\overline{Q} \parr Q\right)}\ \text{Proposition 3}}{\left(\overline{P} \mathbin{\&_p} \overline{Q}\right) \parr \left(P \oplus_p Q\right)}\ \textit{confine}}{\left(\overline{P} \oplus_p \overline{Q}\right) \parr \left(\circ \mathbin{\&_p} \circ\right) \parr \left(P \oplus_p Q\right)}\ \textit{confine}}{\left(\overline{P} \oplus_p \overline{Q}\right) \parr \left(P \oplus_p Q\right)}\ \textit{idempotency}$$

Notice, by instantiating the above with process embeddings, $\vdash [\![t_1 \sqcap t_2]\!] \multimap [\![t_1]\!] \mathbin{\&_p} [\![t_2]\!]$ and $\vdash [\![t_1]\!] \mathbin{\&_p} [\![t_2]\!] \multimap [\![t_1 +_p t_2]\!]$ hold. Hence, by Theorem 2, there is also a proof of the following.

$$\vdash [\![t_1 \sqcap t_2]\!] \multimap [\![t_1 +_p t_2]\!]$$

As guaranteed by Theorem 4, the above linear implication can also be established by probabilistic simulation. For example, process $a \sqcap b$ simulates $a +_p b$. This holds since $\mathcal{R}$ such that $a \mathrel{\mathcal{R}} \mathbf{1}_{a \sqcap b}$, $b \mathrel{\mathcal{R}} \mathbf{1}_{a \sqcap b}$, and $\mathtt{ok} \mathrel{\mathcal{R}} \mathbf{1}_{\mathtt{ok}}$ defines a weak probabilistic simulation such that $[\![a \mathbin{\&_p} b]\!] \mathrel{\hat{\mathcal{R}}} [\![a \sqcap b]\!]$. The converse does not hold since $a \sqcap b \overset{a}{\Longrightarrow} \mathbf{1}_{\mathtt{ok}}$, which is a transition that cannot be matched by distribution $p\mathbf{1}_a + (1-p)\mathbf{1}_b$. Hence, by Theorem 4, the converse implication $P \oplus_p Q \multimap P \mathbin{\&} Q$ also does **not** hold in general.

4.2 Distributivity properties, some forbidden others permitted

We highlight, quite subtly, that we must also forbid certain distributivity properties over parallel composition. Operator $\oplus_p$ forbids refinements that undesirably leak information. For example, processes $(a \parallel c) +_p (b \parallel d)$ and $(a +_p b) \parallel (c +_p d)$ are unrelated by probabilistic simulation. Therefore, by Theorem 4, the following are unrelated by linear implication.

$$(a \otimes c) \oplus_p (b \otimes d) \qquad \text{is unrelated to} \qquad (a \oplus_p b) \otimes (c \oplus_p d)$$

However we should allow other refinements. For example, the semantics of ΔMAV, does admit the following partial distributivity property, preserving all four possible combinations of parallel actions.

$$\vdash (a \oplus_p b) \otimes (c \oplus_q d) \multimap ((a \otimes c) \oplus_q (a \otimes d)) \oplus_p ((b \otimes c) \oplus_q (b \otimes d))$$

The above distributivity property is also respected by probabilistic simulation introduced in Sec. 2. Observe, both $\delta(((a \parallel c) +_q (a \parallel d)) +_p ((b \parallel c) +_q (b \parallel d)))$ and $\delta((a +_p b) \parallel (c +_q d))$ map to the same underlying probability distribution, hence have the same behaviours.

$$pq\mathbf{1}_{a\parallel c} + p(1-q)\mathbf{1}_{a\parallel d} + (1-p)q\mathbf{1}_{b\parallel c} + (1-p)(1-q)\mathbf{1}_{b\parallel d}$$

Indeed, in general, the following implication holds in ΔMAV, establishing how probabilistic choice distributes over parallel composition.

$$\vdash P \otimes (Q \oplus_p R) \multimap (P \otimes Q) \oplus_p (P \otimes R)$$

There are also distributivity properties relating non-deterministic and probabilistic choice [43]. For example we have that $\vdash (P \mathbin{\&} Q) \oplus_p (P \mathbin{\&} R) \multimap P \mathbin{\&} (Q \oplus_p R)$ holds, as established by the following proof.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\circ}{\circ \mathbin{\&} \circ}\ \mathit{tidy}}{(\circ \mathbin{\&_p} \circ) \mathbin{\&} (\circ \mathbin{\&_p} \circ)}\ \mathit{idempotency}}{\left(\left(\overline{P} \parr P\right) \mathbin{\&_p} \left(\overline{P} \parr P\right)\right) \mathbin{\&} \left(\left(\overline{Q} \parr Q\right) \mathbin{\&_p} \left(\overline{R} \parr R\right)\right)}\ \text{by Proposition 3}}{\left(\left(\overline{P} \mathbin{\&_p} \overline{P}\right) \parr (P \oplus_p P)\right) \mathbin{\&} \left(\left(\overline{Q} \mathbin{\&_p} \overline{R}\right) \parr (Q \oplus_p R)\right)}\ \text{by } \mathit{confine}}{\left(\left(\overline{P} \mathbin{\&_p} \overline{P}\right) \parr P\right) \mathbin{\&} \left(\left(\overline{Q} \mathbin{\&_p} \overline{R}\right) \parr (Q \oplus_p R)\right)}\ \mathit{idempotency}}{\left(\left(\left(\overline{P} \oplus \overline{Q}\right) \mathbin{\&_p} \left(\overline{P} \oplus \overline{R}\right)\right) \parr P\right) \mathbin{\&} \left(\left(\left(\overline{P} \oplus \overline{Q}\right) \mathbin{\&_p} \left(\overline{P} \oplus \overline{R}\right)\right) \parr (Q \oplus_p R)\right)}\ \text{by } \mathit{choose}}{\left(\left(\overline{P} \oplus \overline{Q}\right) \mathbin{\&_p} \left(\overline{P} \oplus \overline{R}\right)\right) \parr (P \mathbin{\&} (Q \oplus_p R))}\ \text{by } \mathit{external}$$

By Theorem 4, we have that $(t_1 \sqcap t_2) +_p (t_1 \sqcap t_3)$ simulates $t_1 \sqcap (t_2 +_p t_3)$, for any process. For example, $a \sqcap (b +_p c)$ is simulated by $(a \sqcap b) +_p (a \sqcap c)$. To see why, observe relation $\mathcal{S}$ defined such that $a \sqcap (b +_p c)\ \mathcal{S}\ p\mathbf{1}_{a \sqcap b} + (1-p)\mathbf{1}_{a \sqcap c}$ and $s\ \mathcal{S}\ \mathbf{1}_s$, for any s, is a simulation; for which $[\![a \sqcap (b +_p c)]\!]\ \hat{\mathcal{S}}\ [\![(a \sqcap b) +_p (a \sqcap c)]\!]$.

The converse of the above simulation does not hold. Hence, as a consequence of Theorem 4, the converse of the above implication does not hold in ΔMAV. I.e., in general, the following is **not** provable: $P \with (Q \oplus_p R) \multimap (P \with Q) \oplus_p (P \with R)$.

4.3 But are the medial rules necessary in ΔMAV?

The most mysterious rules of ΔMAV are the *medial* rules. The justification we provide here is purely logical, although these rules are likely to play a more significant role when considering more expressive process calculi with full sequential composition and mixing suitable notions of internal and external choice (sometimes known as angelic/daemonic choices [31]).

Here we show the *medial* rules are necessary in order for cut-elimination to hold. *Medial* rules capture a pattern where a weaker additive distributes over a stronger additive, where $\with < \with_p < \oplus_p < \oplus$. This is a derived property of the standard additives in linear logic; namely the implication $(P \with Q) \oplus (R \with S) \multimap (P \oplus R) \with (Q \oplus S)$ is provable, while its converse does not hold. The corresponding property for the sub-additive is not derivable without the *medials*. Only by including an explicit *medial* rule in Fig. 2 can we prove the following property.

$$(P \with_p Q) \oplus_q (R \with_p S) \multimap (P \oplus_q R) \with_p (Q \oplus_q S)$$

We are forced to include several further *medial* rules, induced by associativity and commutativity. This is more surprising since all other *medial* rules correspond to implications provable without including any *medial* rules. For example, we have the following proof of implication $(P \with Q) \with_q (R \with S) \multimap (P \with_q R) \with (Q \with_q S)$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\circ}{(\circ \with_q \circ) \with (\circ \with_q \circ)}\ \textit{tidy and itempotency}}{\left(\left(\overline{P} \parr P\right) \with_q \left(\overline{R} \parr R\right)\right) \with \left(\left(\overline{Q} \parr Q\right) \with_q \left(\overline{S} \parr S\right)\right)}\ \textit{interact}}{\left(\left(\left(\overline{P} \oplus \overline{Q}\right) \parr P\right) \with_q \left(\left(\overline{R} \oplus \overline{S}\right) \parr R\right)\right) \with \left(\left(\left(\overline{P} \oplus \overline{Q}\right) \parr Q\right) \with_q \left(\left(\overline{R} \oplus \overline{S}\right) \parr S\right)\right)}\ \textit{choose}}{\left(\left(\left(\overline{P} \oplus \overline{Q}\right) \oplus_q \left(\overline{R} \oplus \overline{S}\right)\right) \parr (P \with_q R)\right) \with \left(\left(\left(\overline{P} \oplus \overline{Q}\right) \oplus_q \left(\overline{R} \oplus \overline{S}\right)\right) \parr (Q \with_q S)\right)}\ \textit{confine}}{\left(\left(\overline{P} \oplus \overline{Q}\right) \oplus_q \left(\overline{R} \oplus \overline{S}\right)\right) \parr \left((P \with_q R) \with (Q \with_q S)\right)}\ \textit{external}$$

The above implication does not mean rule $\dfrac{(P \with Q) \with_q (R \with S)}{(P \with_q R) \with (Q \with_q S)}$ is admissible (redundant in ΔMAV). To see why, consider the following observations. Firstly, observe the following is provable without using any medial rules.

$$(a_1 \parr a_2) \with_p ((b_1 \with_q (c \with d)) \parr (b_2 \oplus_q (c \with d))) \multimap (a_1 \with_p (b_1 \with_q (c \with d))) \parr (a_2 \oplus_p (b_2 \oplus_q (c \with d)))$$

Now, assuming $r = (1-p)q$ and $p = s(1-r)$, observe the following are equivalent by associativity and commutativity of the sub-additives.

$$(a_1 \with_p (b_1 \with_q (c \with d))) \parr (a_2 \oplus_p (b_2 \oplus_q (c \with d))) \equiv (b_1 \with_r (a_1 \with_s (c \with d))) \parr (b_2 \oplus_r (a_2 \oplus_s (c \with d)))$$

Thirdly, observe the following implication is provable, without any medial rules.

$$\begin{aligned}&(b_1 \with_r (a_1 \with_s (c \with d))) \parr (b_2 \oplus_r (a_2 \oplus_s (c \with d)))\\ &\qquad \multimap (b_1 \with_r ((a_1 \with_s c) \with (a_1 \with_s d))) \parr (b_2 \oplus_r ((a_2 \oplus_s c) \with (a_2 \oplus_s d)))\end{aligned}$$

Now, assuming cut elimination holds, combining the above three observations, necessarily, we can construct a cut-free proof of the following implication.

$$(a_1 \parr a_2) \mathbin{\&_p} ((b_1 \mathbin{\&_q} (c \mathbin{\&} d)) \parr (b_2 \oplus_q (c \mathbin{\&} d)))$$
$$\multimap (b_1 \mathbin{\&_r} ((a_1 \mathbin{\&_s} c) \mathbin{\&} (a_1 \mathbin{\&_s} d))) \parr (b_2 \oplus_r ((a_2 \oplus_s c) \mathbin{\&} (a_2 \oplus_s d)))$$

Unfortunately, the above implication is not provable without medial rules. Specifically, we require medial rules commuting the sub-additives over *with* in order to establish the proof of the above implication. This example is extracted from exactly where the cut elimination would fail if the medial rules are omitted. Thus the medial rules are not a design decision, but necessary in order for cut-elimination to hold.

5 On the proof of cut-elimination (Theorem 2)

Proving proof normalisation results involves extensive case analysis; hence we provide only a sketch proof of cut elimination proof for ΔMAV. The interesting point is that the idempotency of sub-additives is problematic, giving rise to infinite derivations. For example, formula $a \oplus b$ has infinitely many premises, including those of the form $a \mathbin{\&_{1/2 - 1/2^n}} (a \oplus b)$.

To handle such problems caused by idempotency in the cut elimination proof we move to a semantically equivalent but more controlled version of ΔMAV, turning *idempotency*, from an equivalence into the following inference rules.

$$\frac{\mathcal{C}\{\ R \oplus_p R\ \}}{\mathcal{C}\{\ R\ \}}\ \textit{contract} \qquad \frac{\mathcal{C}\{\ \circ\ \}}{\mathcal{C}\{\ \circ \mathbin{\&_p} \circ\ \}}\ \textit{tidy distribution} \qquad \frac{\mathcal{C}\{\ P \mathbin{\&_r} Q\ \}}{\mathcal{C}\{\ P \oplus_r Q\ \}}\ \textit{special case of confine}$$

The proof of cut-elimination (Theorem 2) proceeds by, firstly, observing rule $\frac{P \otimes \overline{P}}{\circ}\ \textit{cut}$ can be broken down to its atomic form *co-interact* using the following *co-rules*.

$$\frac{\mathcal{C}\{\ (P \oplus R) \otimes (Q \mathbin{\&} S)\ \}}{\mathcal{C}\{\ (P \otimes Q) \oplus (R \otimes S)\ \}}\ \text{co-additives} \qquad \frac{\mathcal{C}\{\ (P \oplus_p Q) \otimes (R \mathbin{\&_p} S)\ \}}{\mathcal{C}\{\ (P \otimes R) \oplus_p (Q \otimes S)\ \}}\ \text{co-confine}$$

$$\frac{\mathcal{C}\{\ \circ \oplus \circ\ \}}{\mathcal{C}\{\ \circ\ \}}\ \text{co-tidy} \qquad \frac{\mathcal{C}\{\ a \otimes \overline{a}\ \}}{\mathcal{C}\{\ \circ\ \}}\ \text{co-interact} \qquad \frac{\mathcal{C}\{\ P\ \}}{\mathcal{C}\{\ P \mathbin{\&_p} P\ \}}\ \text{co-contract}$$

$$\frac{\mathcal{C}\{\ (P \sqcap R) \sqcup (Q \sqcap S)\ \}}{\mathcal{C}\{\ (P \sqcup Q) \sqcap (R \sqcup S)\ \}}\ \text{medial} \qquad \text{where } (\sqcap, \sqcup) \in \{(\&_q, \otimes), (\&_q, \oplus), (\oplus_p, \oplus), (\triangleleft, \otimes)\}$$

We then proceed by the following strategy to show all such co-rules can be eliminated. We firstly apply a technique called decomposition [18, 39, 40], showing instances of the problematic *contract* rule can be pushed to the bottom of a proof. This involves introducing further *co-rules*, notably the rule *co-contract*, which is pushed to the top of the proof. The technical challenge with decomposition is devising a measure controlling explosions in the size of the proof, based on the topology of the proof, caused by permuting contractions with co-contractions.

▶ **Lemma 6** (decomposition). *For any derivation* $\frac{S}{P}$*, including co-rules, there exists* Q *and* R *such that there is a derivation:*

$$\begin{array}{l} S \\ \hline R \\ \hline Q \\ \hline P \end{array} \begin{array}{l} \\ \textit{using co-contract only} \\ \textit{including co-rules but without contract or co-contract} \\ \textit{using contract only} \end{array}$$

Notice, when decomposition is applied to a proof, which must have premise $\circ$, the *co-contract* rules disappear, becoming instances of *tidy distribution*. This way, we transform a proof of P into a proof of some formula Q which does not use *contract* or *co-contract* rules, such that Q is reachable from P using only the *contract* rule. For the proof of Q, that does not use *contract* or *co-contract* rules, we can apply a technique called *splitting* [19]. Splitting generalises the effect of applying rules in sequent-like contexts.

▶ **Lemma 7** (splitting). *In the following, killing contexts are multi-hole contexts defined by grammar* $\mathcal{T}\{\ \} ::= \{\ \cdot\ \} \mid \mathcal{T}\{\ \} \mathbin{\&} \mathcal{T}\{\ \}$. *The following hold in* ΔMAV *without contract, but with* tidy distribution *and the special case of confine:*

- *If* $\vdash (P \mathbin{\&} Q) \parr R$ *then* $\vdash P \parr R$ *and* $\vdash Q \parr R$.
- *If* $\vdash (P \mathbin{\&}_p Q) \parr R$, *there exist* U, V *such that* $\dfrac{U \oplus_p V}{R}$ *and both* $\vdash P \parr U$ *and* $\vdash Q \parr V$ *hold.*
- *If* $\vdash (P \oplus_p Q) \parr R$, *there exist* U, V *such that* $\dfrac{U \mathbin{\&}_p V}{R}$ *and both* $\vdash P \parr U$ *and* $\vdash Q \parr V$ *hold.*
- *If* $\vdash (P \triangleleft Q) \parr R$, *there exist* $\mathcal{T}\{\ \}$, U_i *and* V_i *such that* $\dfrac{\mathcal{T}\{\ U_i \triangleleft V\ \}}{R}$ *and, for all* i, *both* $\vdash P \parr U_i$ *and* $\vdash Q \parr V_i$ *hold.*
- *If* $\vdash (P \otimes Q) \parr R$, *there exist* $\mathcal{T}\{\ \}$, U_i *and* V_i *such that* $\dfrac{\mathcal{T}\{\ U_i \parr V_i\ \}}{R}$ *and, for all* i, $\vdash P \parr U_i$ *and* $\vdash Q \parr V_i$.
- *If* $\vdash (P \oplus Q) \parr R$ *then, there exist* W_i *such that* $\dfrac{\mathcal{T}\{\ W_i\ \}}{R}$ *and, for all* i, *either* $\vdash P \parr W_i$ *or* $\vdash Q \parr W_i$ *hold.*
- *If* $\vdash a \parr R$ *then* $\dfrac{\mathcal{T}\{\ \overline{a}\ \}}{R}$.
- *If* $\vdash \overline{a} \parr R$ *then* $\dfrac{\mathcal{T}\{\ a\ \}}{R}$.

Splitting is then used to extended sequent-like contexts to any context.

▶ **Lemma 8** (context reduction). *If, for all* R, $\vdash P \parr R$ *yields* $\vdash Q \parr R$ *then, for all contexts* $\mathcal{C}\{\ \}$, $\vdash \mathcal{C}\{\ P\ \}$ *yields* $\vdash \mathcal{C}\{\ Q\ \}$.

By using splitting and context reduction, the co-rules previously introduced in this section are shown to be admissible, which together show cut is admissible in the fragment without contraction. The first three co-rule elimination lemmas concern only connectives of MALL [20].

▶ **Lemma 9.** *If* $\vdash \mathcal{C}\{\ \circ \oplus \circ\ \}$ *then* $\vdash \mathcal{C}\{\ \circ\ \}$.

▶ **Lemma 10.** *If* $\vdash \mathcal{C}\{\ (P \oplus Q) \otimes (R \mathbin{\&} S)\ \}$ *holds, then it holds that* $\vdash \mathcal{C}\{\ (P \otimes R) \oplus (Q \otimes S)\ \}$.

▶ **Lemma 11.** *If* $\vdash \mathcal{C}\{\ a \otimes \overline{a}\ \}$ *then* $\vdash \mathcal{C}\{\ \circ\ \}$, *for any atom* a.

The following co-rule elimination lemma involves the probabilistic sub-additives.

▶ **Lemma 12.** *If* $\vdash \mathcal{C}\{\ (P \oplus_p Q) \otimes (R \mathbin{\&}_p S)\ \}$ *holds,* $\vdash \mathcal{C}\{\ (P \otimes R) \oplus_p (Q \otimes S)\ \}$ *holds.*

The four extra *medial* rules can also be eliminated.

▶ **Lemma 13.** *For any* $(\sqcap, \sqcup) \in \{(\&_q, \otimes), (\&_q, \oplus), (\oplus_p, \oplus), (\triangleleft, \otimes)\}$, *if* $\vdash \mathcal{C}\{\ (P \sqcap R) \sqcup (Q \sqcap S)\ \}$ *then* $\vdash \mathcal{C}\{\ (P \sqcup Q) \sqcap (R \sqcup S)\ \}$.

We can now establish cut elimination for the proof system described at the beginning of this section, without *idempotency*, but with three inference rules: *contract*, *tidy distribution* and the *special case of confine*. Having applied decomposition (Lemma 6) to push *contract* to the bottom of the proof, the proof combines the above lemmas to remove each *co-rule*. This leaves a system without *co-rules*.

Finally, we obtain our main result (Theorem 2): cut elimination in the more controlled system implies cut elimination in ΔMAV, simply be substituting *contract*, *tidy distribution* and the *special case of confine* with instances of *idempotency* and *confine*.

6 Related work on Sub-Additive Operators and Nominal Quantifiers

Between the standard additives of multiplicative linear logic, *with* and *plus*, there are further sub-additive operators. Roversi [35] proposed a sub-additive operator, say $\boxplus$, also forbidding projection and injection, that is self-dual. Note a self-dual operator is such that the linear negation of $P \boxplus Q$ is $\overline{P} \boxplus \overline{Q}$, i.e., the operator is de Morgan dual to itself.

Such a self-dual sub-additive operator cannot be used to model probabilistic choice in the processes-as-formulae paradigm. The problem is the following implication is provable $(a \boxplus b) \otimes (c \boxplus d) \multimap (a \otimes c) \boxplus (b \otimes d)$. Consequently, self-dual sub-additives are unsound with respect to probabilistic simulation (notice the possibility of $a \otimes d$ or $b \otimes c$ occurring has been excluded in the formula on the right). The pair of probabilistic sub-additives $\&_p$ and $\oplus_p$, were discovered by seeking more controlled variants of $\boxplus$ such that the above unsound distributivity property is **forbidden**.

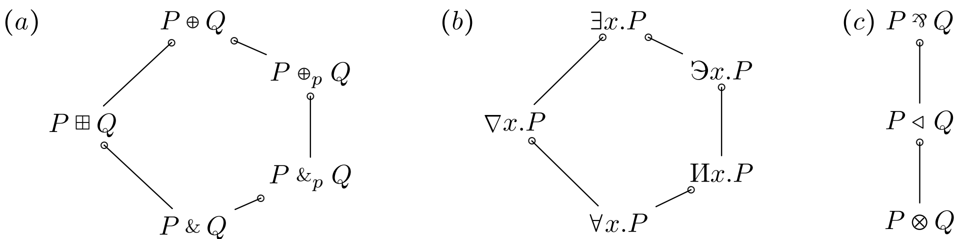


Figure 3 Relationships between various operators in extensions of linear logic: (a) the additives and sub-additives, (b) the first-order quantifiers and nominal quantifiers, (c) the multiplicatives.

Figure 3(a) compares additives $\&$, $\&_p$, $\oplus_p$, $\oplus$ and $\boxplus$. Notice similarities with Fig 3(b) depicting de Morgan dual pair of nominal quantifiers, $\text{И}x.P$ and $\ni x.P$, located between *for all* and *exists* [23]. Similarly, to the sub-additives, the justification for the pair of nominal quantifiers, rather than a self-dual nominal quantifier [14, 34, 35], say $\nabla x.P$, was to soundly model private names in direct logical embeddings of π-calculus processes [32].

Related work at the intersection of linear logic and probabilistic programs is typically denotational (of a model theoretic flavour). For example, *probabilistic coherence spaces* [16, 12] provide a *probabilistic denotational semantics* [26, 7] for linear logic but with standard additives *with* and *plus* only. Probabilistic coherence spaces and related models are typically used directly to provide a semantics for functional probabilistic programming languages, such as PCF with random number generators [13, 6] or a probabilistic λ-calculus [29]. However, probabilistic extensions of linear logic itself, giving rise to probabilistic sub-additives sound with respect the probabilistic choice in process calculi, have not previously been investigated.

7 Conclusion

This paper exposes an extended *syntax* and proof system for linear logic with explicit probabilistic choice operators. The rules for these *sub-additives* are determined by studying a generalisation of *cut elimination* (Theorem 2), leaving no room for design decisions. When designing process preorders, we are confronted by a vast design space. Thus $\Delta\mathsf{MAV}$ (Fig. 2) can assist objectively with resolving language design decisions. I argue linear implication is a compelling notion of probabilistic refinement, being sound with respect to *weak (complete) probabilistic simulation* (Theorem 4), hence also *probabilistic may testing*. Furthermore, linear implication has the advantage that it is the coarsest notion of refinement for probabilistic concurrent processes in the literature respecting action refinement (Corollary 5).

Interestingly, the proof of cut elimination demands a technique called *decomposition*, Lemma 6, to handle idempotency of choice, which, previously, has only been *necessary* for handling modalities in non-commutative logic NEL [39, 19]. Details of the proof theory are reserved for an extended version.

Future work includes explaining the connections between the quantitative modal logics, such as the quantitative modal μ-calculus [25], and $\Delta\mathsf{MAV}$. Future work may also consider richer process models in $\Delta\mathsf{MAV}$ and its extensions [24]. For example, by using positive and negative atoms to model inputs and outputs [3, 22], we can model probabilistic calculi with communication. A related question is whether the operator $\&_p$ is useful when modelling processes. Recall $\&_p$ was discovered, synthetically, as the operator de Morgan dual to probabilistic choice $\oplus_p$. To help understand the nature of $\&_p$, observe that it is related to $\oplus_p$ in a similar fashion that, in the internal π-calculus [36], fresh name binding ν is related to internal input (which receives a name, but only if it is fresh). By using this analogy, $\&_p$ can model branches of an input that preserves a probability distribution by using knowledge of the probability distribution over branches with which it interacts (perhaps by measuring previous interactions with a controller, for example), and only interacts if the distribution matches the criteria specified by the internal choice (as suggested by rule *confine*). Such constraints could be useful for preventing systems from being composed whenever the random behaviour of one component falls out of expected bounds of another component (possibly causing a component that receives messages on a random channel to fail to meet its specification). Considering possible connections between $\&_p/\oplus_p$ and angelic/daemonic probabilistic choices [31] is also future work. To help the reader digest this novel theory, initially, only simple and indisputable core process models are discussed in the current paper.

References

1 Samy Abbes and Albert Benveniste. True-concurrency probabilistic models: Branching cells and distributed probabilities for event structures. *Information and Computation*, 204(2):231–274, 2006. doi:10.1016/j.ic.2005.10.001.

2 Christel Baier and Holger Hermanns. Weak bisimulation for fully probabilistic processes. In *Computer Aided Verification*, pages 119–130. Springer, 1997. doi:10.1007/3-540-63166-6_14.

3 Paola Bruscoli. A Purely Logical Account of Sequentiality in Proof Search. In *ICLP*, pages 302–316, 2002. doi:10.1007/3-540-45619-8_21.

4 Kaustuv Chaudhuri, Nicolas Guenot, and Lutz Straßburger. The focused calculus of structures. In *CSL*, volume 12, pages 159–173, 2011. doi:10.4230/LIPIcs.CSL.2011.159.

5 Gabriel Ciobanu and Ross Horne. Behavioural Analysis of Sessions using the Calculus of Structures. In *PSI 2015, 25-27 August, Kazan, Russia*, volume 9609 of *LNCS*, pages 91–106, 2015. doi:10.1007/978-3-319-41579-6_8.

6. Ugo Dal Lago, Claudia Faggian, Benoît Valiron, and Akira Yoshimizu. The Geometry of Parallelism: Classical, Probabilistic, and Quantum Effects. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL 2017, pages 833–845. ACM, 2017. doi:10.1145/3009837.3009859.
7. Vincent Danos and Russell S. Harmer. Probabilistic Game Semantics. *ACM Trans. Comp. Logic*, 3(3):359–382, July 2002. doi:10.1145/507382.507385.
8. Jerry den Hartog, Erik P. de Vink, and Jacobus W. De Bakker. Metric semantics and full abstractness for action refinement and probabilistic choice. *Electronic Notes in Theoretical Computer Science*, 40:72–99, 2001. doi:10.1016/S1571-0661(05)80038-6.
9. Yuxin Deng. *Semantics of Probabilistic Processes: An Operational Approach*. Springer, 2015. doi:10.1007/978-3-662-45198-4.
10. Yuxin Deng, Matthew Hennessy, Rob van Glabbeek, and Carroll Morgan. Characterising Testing Preorders for Finite Probabilistic Processes. In *22nd Annual IEEE Symposium on Logic in Computer Science (LICS 2007)*, pages 313–325, July 2007. doi:10.1109/LICS.2007.15.
11. Yuxin Deng, Rob van Glabbeek, Matthew Hennessy, and Carroll Morgan. Characterising Testing Preorders for Finite Probabilistic Processes. *Logical Methods in Computer Science*, 4(4), 2008. doi:10.2168/LMCS-4(4:4)2008.
12. Thomas Ehrhard, Michele Pagani, and Christine Tasson. The computational meaning of probabilistic coherence spaces. In *LICS*, pages 87–96. IEEE, 2011. doi:10.1109/LICS.2011.29.
13. Thomas Ehrhard, Christine Tasson, and Michele Pagani. Probabilistic coherence spaces are fully abstract for probabilistic PCF. *In Proc. POPL*, 49(1):309–320, 2014. doi:10.1145/2535838.2535865.
14. Andrew Gacek, Dale Miller, and Gopalan Nadathur. Nominal abstraction. *Information and Computation*, 209(1):48–73, 2011. doi:10.1016/j.ic.2010.09.004.
15. Jean-Yves Girard. Linear Logic. *Theoretical Computer Science*, 50(1):1–112, 1987. doi:10.1016/0304-3975(87)90045-4.
16. Jean-Yves Girard. Between logic and quantic: a tract. *Linear logic in computer science*, 316:346–381, 2004. doi:10.1017/CBO9780511550850.011.
17. Alessio Guglielmi. A system of interaction and structure. *ACM Transactions on Computational Logic*, 8(1), 2007. doi:10.1145/1182613.1182614.
18. Alessio Guglielmi and Tom Gundersen. Normalisation Control in Deep Inference via Atomic Flows. *Logical Methods in Computer Science*, 4(1), 2008. doi:10.2168/LMCS-4(1:9)2008.
19. Alessio Guglielmi and Lutz Straßburger. A system of interaction and structure V: the exponentials and splitting. *Mathematical Structures in Computer Science*, 21(3):563–584, 2011. doi:10.1017/S096012951100003X.
20. Ross Horne. The Consistency and Complexity of Multiplicative Additive System Virtual. *Sci. Ann. Comp. Sci.*, 25(2):245–316, 2015. doi:10.7561/SACS.2015.2.245.
21. Ross Horne, Sjouke Mauw, and Alwen Tiu. Semantics for Specialising Attack Trees based on Linear Logic. *Fundamenta Informaticae*, 153(1-2):57–86, 2017. doi:10.3233/FI-2017-1531.
22. Ross Horne and Alwen Tiu. Constructing Weak Simulations from Linear Implications for Processes with Private Names. *Mathematical Structure in Computer Science*, pages 1–34, 2019. doi:10.1017/S0960129518000452.
23. Ross Horne, Alwen Tiu, Bogdan Aman, and Gabriel Ciobanu. Private Names in Non-Commutative Logic. In *CONCUR 2016*, pages 31:1–31:16. LIPIcs, 2016. doi:10.4230/LIPIcs.CONCUR.2016.31.
24. Ross Horne, Alwen Tiu, Bogdan Aman, and Gabriel Ciobanu. De Morgan Dual Nominal Quantifiers Modelling Private Names in Non-Commutative Logic. *ACM Transactions on Computational Logic (TOCL)*, 20(4), 2019.
25. Michael Huth and Marta Z. Kwiatkowska. Quantitative analysis and model checking. In *Proceedings of Twelfth Annual IEEE Symposium on Logic in Computer Science*, pages 111–122, 1997. doi:10.1109/LICS.1997.614940.

26 Cliff Jones and Gordon Plotkin. A probabilistic powerdomain of evaluations. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science*, pages 186–195, June 1989. doi:10.1109/LICS.1989.39173.

27 Bengt Jonsson and Wang Yi. Compositional testing preorders for probabilistic processes. In *LICS*, pages 431–441. IEEE, 1995. doi:10.1109/LICS.1995.523277.

28 Bengt Jonsson, Wang Yi, and Kim G Larsen. Probabilistic extensions of process algebras. *Handbook of process algebra*, pages 685–710, 2001. doi:10.1016/B978-044482830-9/50029-1.

29 Ugo Dal Lago and Margherita Zorzi. Probabilistic operational semantics for the lambda calculus. *RAIRO - Theoretical Informatics and Applications*, 46(3):413–450, 2012. doi:10.1051/ita/2012012.

30 Kim G Larsen and Arne Skou. Bisimulation through probabilistic testing. *Information and computation*, 94(1):1–28, 1991. doi:10.1016/0890-5401(91)90030-6.

31 Annabelle McIver and Carroll Morgan. *Abstraction, Refinement And Proof For Probabilistic Systems (Monographs in Computer Science)*. SpringerVerlag, 2004. doi:10.1007/b138392.

32 Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100(1):1–77, 1992. doi:10.1016/0890-5401(92)90008-4.

33 Catuscia Palamidessi and Oltea Mihaela Herescu. A randomized encoding of the π-calculus with mixed choice. *Theoretical Computer Science*, 335(2-3):373–404, 2005. doi:10.1016/j.tcs.2004.11.020.

34 Andrew Pitts. Nominal Logic, a First Order Theory of Names and Binding. *Information and Computation*, 186(2), 2003. doi:10.1016/S0890-5401(03)00138-X.

35 Luca Roversi. A deep inference system with a self-dual binder which is complete for linear lambda calculus. *J. of Logic and Computaion*, 26(2):677–698, 2016. doi:10.1093/logcom/exu033.

36 Davide Sangiorgi. pi-Calculus, Internal Mobility, and Agent-Passing Calculi. *Theor. Comput. Sci.*, 167(1&2):235–274, 1996. doi:10.1016/0304-3975(96)00075-8.

37 Roberto Segala and Nancy Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995. doi:10.1007/3-540-56454-3_13.

38 Lutz Straßburger. A local system for linear logic. In *LPAR*, volume 2514, pages 388–402. Springer, 2002. doi:10.1007/3-540-36078-6_26.

39 Lutz Straßburger and Alessio Guglielmi. A system of interaction and structure IV: The exponentials and decomposition. *ACM T. Comp. Log.*, 12(4):23:1–39, 2011. doi:10.1145/1970398.1970399.

40 Andrea Aler Tubella, Alessio Guglielmi, and Benjamin Ralph. Removing Cycles from Proofs. In *CSL*, pages 9:1–9:17, 2017. doi:10.4230/LIPIcs.CSL.2017.9.

41 Rob van Glabbeek and Ursula Goltz. Refinement of actions and equivalence notions for concurrent systems. *Acta Informatica*, 37(4-5):229–327, 2001. doi:10.1007/s002360000041.

42 Daniele Varacca, Hagen Völzer, and Glynn Winskel. Probabilistic event structures and domains. In *CONCUR*, pages 481–496, 2004. doi:10.1007/978-3-540-28644-8_31.

43 Daniele Varacca and Glynn Winskel. Distributing probability over non-determinism. *Mathematical Structures in Computer Science*, 16(1):87–113, 2006. doi:10.1017/S0960129505005074.

A Lower Bound of the Number of Rewrite Rules Obtained by Homological Methods

Mirai Ikebuchi
Massachusetts Institute of Technology, Computer Science and Artificial Intelligence Laboratory, Cambridge, USA
ikebuchi@mit.edu

Abstract

It is well-known that some equational theories such as groups or boolean algebras can be defined by fewer equational axioms than the original axioms. However, it is not easy to determine if a given set of axioms is the smallest or not. Malbos and Mimram investigated a general method to find a lower bound of the cardinality of the set of equational axioms (or rewrite rules) that is equivalent to a given equational theory (or term rewriting systems), using homological algebra. Their method is an analog of Squier's homology theory on string rewriting systems. In this paper, we develop the homology theory for term rewriting systems more and provide a better lower bound under a stronger notion of equivalence than their equivalence. The author also implemented a program to compute the lower bounds.

2012 ACM Subject Classification Theory of computation → Rewrite systems

Keywords and phrases Term rewriting systems, Equational logic, Homological algebra

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.24

1 Introduction

The purpose of this paper is to find a lower bound of the number of axioms that are equivalent to a given equational theory. For example, the theory of groups is given by the following axioms:

$$\begin{array}{lll} G_1.\ m(m(x_1,x_2),x_3)=m(x_1,m(x_2,x_3)), & G_2.\ m(x_1,e)=x_1, & G_3\ m(e,x_1)=x_1, \\ G_4.\ m(i(x_1),x_1)=e, & G_5.\ m(x_1,i(x_1))=e. & \end{array} \tag{1}$$

It is well-known that G_2 and G_5 can be derived from only $\{G_1, G_3, G_4\}$. Moreover, the theory of groups can be given by two axioms: the axiom

$$m(x_1, i(m(m(i(m(i(x_2), m(i(x_1), x_3))), x_4), i(m(x_2, x_4))))) = x_3$$

together with G_4 is equivalent to the group axioms [4]. If we use the new symbol n which corresponds to the "multiplication of inverses" $m(i(x_1), i(x_2))$, a single axiom,

$$n(x_1, n(n(n(e, x_2), n(n(n(e, x_3), x_3), x_4)), n(n(e, x_1), x_2))) = x_4,$$

is equivalent to the group axioms [5]. However, no single axiom written in symbols m, i, e is equivalent to the group axioms. This is stated without proof by Tarski [9] and published proofs are given by Neumann [4] and Kunen [2]. Malbos and Mimram developed a general method to calculate a lower bound of the number of axioms that are "Tietze-equivalent" to a given complete term rewriting system (TRS) [3, Proposition 23]. We omit the definition of Tietze equivalence here, but roughly speaking, it is an equivalence between equational theories (or TRSs) (Σ_1, R_1), (Σ_2, R_2) where signatures Σ_1 and Σ_2 are not necessarily equal to each other, while the usual equivalence between TRSs is defined for two TRSs (Σ, R_1), (Σ, R_2) over the same signature (specifically, by $\overset{*}{\leftrightarrow}_{R_1} = \overset{*}{\leftrightarrow}_{R_2}$).

© Mirai Ikebuchi;
licensed under Creative Commons License CC-BY
4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019).
Editor: Herman Geuvers; Article No. 24; pp. 24:1–24:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper, we will develop Malbos and Mimram's theory more, and show an inequality which gives a better lower bound of the number of axioms with respect to the usual equivalence between TRSs over the same signature. For the theory of groups, our inequality gives that the number of axioms equivalent to the group axioms is greater than or equal to 2, so we have another proof of Tarski's theorem above as a special case. Our lower bound is algorithmically computable if a complete TRS is given.

We will first give the statement of our main theorem and some examples in Section 2. Then, we will see Malbos-Mimram's work briefly. The idea of their work is to provide an algebraic structure to TRSs and extract information of the TRSs, called homology groups, which are invariant under Tietze equivalence. The basics of such algebraic tools are given in Section 3, and we will see the idea of the construction of the homology groups of TRSs in Section 4. Finally, in Section 5, we will prove our main theorem.

2 Main Theorem

In this section, we will see our main theorem and some examples. Throughout this paper, we assume that any terms are over the set of variables $\{x_1, x_2, \dots\}$ and all signatures we consider are unsorted. For a signature Σ, let $T(\Sigma)$ denote the set of terms over the signature Σ and the set of variables $\{x_1, x_2, \dots\}$.

▶ **Definition 1.** *Let (Σ, R) be a TRS. The degree of R, denoted by $\deg(R)$, is defined by*

$$\deg(R) = \gcd\{\#_i l - \#_i r \mid l \to r \in R, i = 1, 2, \dots\}$$

where $\#_i t$ is the number of occurrences of x_i in t for $t \in T(\Sigma)$ and we define $\gcd\{0\} = 0$ for convenience. For example, $\deg(\{f(x_1, x_2, x_2) \to x_1,\ g(x_1, x_1, x_1) \to e\}) = \gcd\{0, 2, 3\} = 1$.

Let $(\Sigma, R = \{l_1 \to r_1, \dots, l_n \to r_n\})$ be a TRS and $\mathrm{CP}(R) = \{(t_1, s_1), \dots, (t_m, s_m)\}$ be the set of the critical pairs of R. For any $i \in \{1, \dots, m\}$, let a_i, b_i be the numbers in $\{1, \dots, n\}$ such that the critical pair (t_i, s_i) is obtained by $l_{a_i} \to r_{a_i}$ and $l_{b_i} \to r_{b_i}$, that is, $t_i = r_{a_i}\sigma \leftarrow l_{a_i}\sigma = C[l_{b_i}\sigma] \to C[r_{b_i}\sigma] = s_i$ for some substitution σ and single-hole context C. Suppose R is complete. We fix an arbitrary rewriting strategy and for a term t, let $\mathrm{nr}_j(t)$ be the number of times $l_j \to r_j$ is used to reduce t into its R-normal form with respect to the strategy. To state our main theorem, we introduce a matrix $D(R)$ and a number $e(R)$:

▶ **Definition 2.** *Suppose $d = \deg(R)$ is prime or 0. If $d = 0$, let $\mathfrak{R}$ be $\mathbb{Z}$, and if d is prime, let $\mathfrak{R}$ be $\mathbb{Z}/d\mathbb{Z}$ (integers modulo d). For $1 \le i \le m$, $1 \le j \le n$, let $D(R)_{ij}$ be the integer $\mathrm{nr}_j(s_i) - \mathrm{nr}_j(t_i) + \delta(b_i, j) - \delta(a_i, j)$ where $\delta(x, y)$ is the Kronecker delta. The matrix $D(R)$ is defined by $D(R) = (D(R)_{ij})_{i=1,\dots,m, j=1,\dots,n}$.*

▶ **Definition 3.** *Let $\mathfrak{R}$ be $\mathbb{Z}$ or $\mathbb{Z}/p\mathbb{Z}$ for any prime p. If an $m \times n$ matrix M over $\mathfrak{R}$ is of the form*

$$\begin{pmatrix}
e_1 & 0 & \dots & \dots & \dots & \dots & \dots & 0 \\
0 & e_2 & 0 & \dots & \dots & \dots & \dots & 0 \\
\vdots & 0 & \ddots & 0 & \dots & \dots & \dots & \vdots \\
\vdots & \vdots & 0 & e_r & 0 & \dots & \dots & \vdots \\
\vdots & \vdots & \vdots & 0 & 0 & \dots & \dots & \vdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \dots & \vdots \\
0 & 0 & \dots & \dots & \dots & \dots & \dots & 0
\end{pmatrix}$$

and e_i divides e_{i+1} for every $1 \le i < r$, we say M is in Smith normal form*. We call e_is the* elementary divisors*.*

It is known that every matrix over $\mathfrak{R}$ can be transformed into Smith normal form by elementary row/column operations, that is, (1) switching a row/column with another row/column, (2) multiplying each entry in a row/column by an invertible element in $\mathfrak{R}$, and (3) adding a multiple of a row/column to another row/column [7, 9.4]. (If $d = 0$, the invertible elements in $\mathfrak{R} \cong \mathbb{Z}$ are 1 and -1, and if d is prime, any nonzero elements in $\mathfrak{R} = \mathbb{Z}/d\mathbb{Z}$ are invertible.) In general, the same fact holds for any principal ideal domain $\mathfrak{R}$.

▶ **Definition 4.** *We define $e(R)$ as the number of invertible elements in the Smith normal form of the matrix $D(R)$ over $\mathfrak{R}$.*

Note that if $\mathfrak{R} = \mathbb{Z}/d\mathbb{Z}$ for a prime d, $e(R)$ is equal to the rank of $D(R)$ since every nonzero elements in $\mathbb{Z}/d\mathbb{Z}$ is invertible.

We state the main theorem.

▶ **Theorem 5.** *Let (Σ, R) be a complete TRS and suppose $d = \deg(R)$ is 0 or prime. For any set of rules R' equivalent to R, i.e., $\overset{*}{\leftrightarrow}_{R'} = \overset{*}{\leftrightarrow}_R$, we have*

$$\#R' \geq \#R - e(R). \tag{2}$$

We shall see some examples.

▶ **Example 6.** Consider the signature $\Sigma = \{0^{(0)}, \mathsf{s}^{(1)}, \mathsf{ave}^{(2)}\}$ and the set R of rules

$A_1.\mathsf{ave}(0,0) \to 0$, $A_2.\mathsf{ave}(x_1, \mathsf{s}(x_2)) \to \mathsf{ave}(\mathsf{s}(x_1), x_2)$, $A_3.\mathsf{ave}(\mathsf{s}(0), 0) \to 0$,
$A_4.\mathsf{ave}(\mathsf{s}(\mathsf{s}(0)), 0) \to s(0)$, $A_5.\mathsf{ave}(\mathsf{s}(\mathsf{s}(\mathsf{s}(x_1))), x_2) \to \mathsf{s}(\mathsf{ave}(\mathsf{s}(x_1), x_2))$.

R satisfies $\deg(R) = 0$ and has one critical pair C:

$\mathsf{ave}(\mathsf{s}(\mathsf{s}(\mathsf{s}(x_1))), \mathsf{s}(x_2))$
$\swarrow A_2$ $\quad$ $A_5 \searrow$
$\mathsf{ave}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(x_1)))), x_2)$ $\quad$ $\mathsf{s}(\mathsf{ave}(\mathsf{s}(x_1), \mathsf{s}(x_2)))$
$A_5 \searrow$ $\quad$ $\swarrow A_2$
$\mathsf{s}(\mathsf{ave}(\mathsf{s}(\mathsf{s}(x_1)), x_2))$

We can see the matrix $D(R)$ is the 5×1 zero matrix. The zero matrix is already in Smith normal form and $e(R) = 0$. Thus, for any R' equivalent to R, $\#R' \geq \#R = 5$. This means there is no smaller TRS equivalent to R. Also, Malbos-Mimram's lower bound, denoted by $s(H_2(\Sigma, R))$, is equal to 3, though we do not explain how to compute it in this paper. (We will briefly see the meaning of $s(H_2(\Sigma, R))$ in Section 4.)

▶ **Example 7.** We compute the lower bound for the theory of groups, (1). A complete TRS R for the theory of groups is given by

$G_1.\ m(m(x_1, x_2), x_3) \to m(x_1, m(x_2, x_3))$ $\quad$ $G_2.\ m(e, x_1) \to x_1$
$G_3.\ m(x_1, e) \to x_1$ $\quad$ $G_4.\ m(x_1, i(x_1)) \to e$
$G_5.\ m(i(x_1), x_1) \to e$ $\quad$ $G_6.\ m(i(x_1), m(x_1, x_2)) \to x_2$
$G_7.\ i(e) \to e$ $\quad$ $G_8.\ i(i(x_1)) \to x_1$
$G_9.\ m(x_1, m(i(x_1), x_2)) \to x_2$ $\quad$ $G_{10}.\ i(m(x_1, x_2)) \to m(i(x_2), i(x_1))$.

Since $\deg(R) = 2$, we set $\mathfrak{R} = \mathbb{Z}/2\mathbb{Z}$. R has 48 critical pairs and we get the 10×48 matrix $D(R)$ given in the appendix. The author implemented a program which takes a complete TRS as input and computes its critical pairs, the matrix $D(R)$, and $e(R)$. The program is available at `https://github.com/mir-ikbch/homtrs`. The author checked $e(R) = \mathrm{rank}(D(R)) = 8$ by the program, and also by MATLAB's `gfrank` function (`https://www.mathworks.com/help/comm/ref/gfrank.html`). Therefore we have $\#R - e(R) = 2$. This provides a new proof that there is no single axiom equivalent to the theory of groups.

Malbos-Mimram's lower bound is given by $s(H_2(\Sigma, R)) = 0$.

C_1: $-(-f(x_1)) \xrightarrow{A_2} -(f(-x_1)) \xrightarrow{A_2} f(-(-x_1)) \xrightarrow{A_1} f(x_1)$; $-(-f(x_1)) \xrightarrow{A_1} f(x_1)$

C_2: $-(-(-x_1)) \xrightarrow{A_1} -x_1$; $-(-(-x_1)) \xrightarrow{A_1} -x_1$

C_3: $-(-(x_1+x_2)) \xrightarrow{A_3} -((-x_1)\cdot(-x_2)) \xrightarrow{A_4} (-(-x_1))+(-(-x_2)) \xrightarrow{A_1} x_1+(-(-x_2)) \xrightarrow{A_1} x_1+x_2$; $-(-(x_1+x_2)) \xrightarrow{A_1} x_1+x_2$

C_4: $-(-(x_1\cdot x_2)) \xrightarrow{A_4} -((-x_1)+(-x_2)) \xrightarrow{A_3} (-(-x_1))\cdot(-(-x_2)) \xrightarrow{A_1} x_1\cdot(-(-x_2)) \xrightarrow{A_1} x_1\cdot x_2$; $-(-(x_1\cdot x_2)) \xrightarrow{A_1} x_1\cdot x_2$

Figure 1 The critical pairs of R.

▶ **Example 8.** Let $\Sigma = \{-^{(1)}, f^{(1)}, +^{(2)}, \cdot^{(2)}\}$ and R be

$$A_1.\ -(-x_1) \to x_1, \qquad A_2.\ -f(x_1) \to f(-x_1),$$
$$A_3.\ -(x_1+x_2) \to (-x_1)\cdot(-x_2), \qquad A_4.\ -(x_1\cdot x_2) \to (-x_1)+(-x_2).$$

We have $\deg(R) = 0$ and R has four critical pairs (Figure 1). The corresponding matrix $D(R)$ and its Smith normal form are computed as

$$D(R) = \begin{pmatrix} 0&0&1&1\\ 2&0&0&0\\ 0&0&1&1\\ 0&0&1&1 \end{pmatrix} \rightsquigarrow \begin{pmatrix} 0&0&1&1\\ 2&0&0&0\\ 0&0&0&0\\ 0&0&0&0 \end{pmatrix} \rightsquigarrow \begin{pmatrix} 0&0&1&0\\ 2&0&0&0\\ 0&0&0&0\\ 0&0&0&0 \end{pmatrix} \rightsquigarrow \begin{pmatrix} 1&0&0&0\\ 0&2&0&0\\ 0&0&0&0\\ 0&0&0&0 \end{pmatrix}.$$

Thus, $\#R - e(R) = 3$. This tells R does not have any equivalent TRS with 2 or fewer rules, and it is not difficult to see R has an equivalent TRS with 3 rules, $\{A_1, A_2, A_3\}$.

Malbos-Mimram's lower bound for this TRS is given by $s(H_2(\Sigma, R)) = 1$.

Although the equality of (2) is attained for the above three examples, it is not guaranteed the equality is attained by some TRS R' in general.

3 Preliminaries on Algebra

In this section, we give a brief introduction to module theory, homological algebra, and Squier's theory of homological algebra for string rewriting systems (SRSs) [8]. Even though Squier's theory is not directly needed to prove our theorem, it is helpful to understand the homology theory for TRSs, which is more complicated than SRSs' case.

3.1 Modules and Homological Algebra

We give basic definitions and theorems on module theory and homological algebra without proofs. For more details, readers are referred to [7, 6] for example.

Modules are the generalization of vector spaces in which the set of scalars form a ring, not necessarily a field.

▶ **Definition 9.** *Let $\mathfrak{R}$ be a ring and $(M,+)$ be an abelian group. For a map $\cdot : R \times M \to M$, $(M,+,\cdot)$ is a* left $\mathfrak{R}$-module *if for all $r, s \in R$ and $x, y \in M$, we have*

$$r\cdot(x+y) = r\cdot x + r\cdot y, \quad (r+s)\cdot x = r\cdot x + s\cdot x, \quad (rs)\cdot x = r\cdot(s\cdot x)$$

where rs denotes the multiplication of r and s in $\mathfrak{R}$. We call the map $\cdot$ scalar multiplication*.*

For a map $\cdot : M \times R \to M$, $(M, +, \cdot)$ is a right $\mathfrak{R}$-module *if for any $r, s \in \mathfrak{R}$ and $x, y \in M$,*

$$(x + y) \cdot r = x \cdot r + y \cdot r, \; x \cdot (r + s) = x \cdot r + x \cdot s, \; x \cdot (sr) = (x \cdot s) \cdot r.$$

If ring $\mathfrak{R}$ is commutative, we do not distinguish between left $\mathfrak{R}$-modules and right $\mathfrak{R}$-modules and simply call them $\mathfrak{R}$-modules.

Linear maps and isomorphisms of modules are also defined in the same way as for vector spaces.

▶ **Definition 10.** *For two left $\mathfrak{R}$-modules $(M_1, +_1, \cdot_1), (M_2, +_2, \cdot_2)$, a group homomorphism $f : (M_1, +_1) \to (M_2, +_2)$ is an* $\mathfrak{R}$-linear map *if it satisfies $f(r \cdot_1 x) = r \cdot_2 f(x)$ for any $r \in \mathfrak{R}$ and $x \in M_1$. An $\mathfrak{R}$-linear map f is an* isomorphism *if it is bijective, and two modules are called* isomorphic *if there exists an isomorphism between them.*

▶ **Example 11.** Any abelian group $(M, +)$ is a $\mathbb{Z}$-module under the scalar multiplication $n \cdot x = \underbrace{x + \cdots + x}_{n}$.

▶ **Example 12.** For any ring $\mathfrak{R}$, the direct product $\mathfrak{R}^n = \underbrace{\mathfrak{R} \times \cdots \times \mathfrak{R}}_{n}$ forms a left $\mathfrak{R}$-module under the scalar multiplication $r \cdot (r_1, \ldots, r_n) = (rr_1, \ldots, rr_n)$.

▶ **Example 13.** Let $\mathfrak{R}$ be a ring and X be a set. $\mathfrak{R}\underline{X}$ denotes the set of formal linear combinations

$$\sum_{x \in X} r_x \underline{x} \quad (r_x \in \mathfrak{R})$$

where $r_x = 0$ except for finitely many xs. The underline is added to emphasize a distinction between $r \in \mathfrak{R}$ and $x \in X$. $\mathfrak{R}\underline{X}$ forms a left $\mathfrak{R}$-module under the addition and the scalar multiplication defined by

$$\left(\sum_{x \in X} r_x \underline{x} \right) + \left(\sum_{x \in X} s_x \underline{x} \right) = \sum_{x \in X} (r_x + s_x) \underline{x}, \quad s \cdot \left(\sum_{x \in X} r_x \underline{x} \right) = \sum_{x \in X} (s r_x) \underline{x}.$$

If X is the empty set, $\mathfrak{R}\underline{X}$ is the left $\mathfrak{R}$-module $\{0\}$ consisting of only the identity element. We simply write 0 for $\{0\}$. $\mathfrak{R}\underline{X}$ is called the *free left $\mathfrak{R}$-module generated by X*. If $\#X = n \in \mathbb{N}$, $\mathfrak{R}\underline{X}$ can be identified with $\mathfrak{R}^n$.

A left $\mathfrak{R}$-module M is said *free* if M is isomorphic to $\mathfrak{R}\underline{X}$ for some X. Free modules have some similar properties to vector spaces. If a left $\mathfrak{R}$-module F if free, there exists a basis (i.e., a subset that is linearly independent and generating) of F. If a free left $\mathfrak{R}$-module F has a basis $(v_1, \ldots, v_n)$, any $\mathfrak{R}$-linear map $f : F \to M$ is uniquely determined if the values $f(v_1), \ldots, f(v_n)$ are specified. Suppose F_1, F_2 are free left $\mathfrak{R}$-modules and $f : F_1 \to F_2$ is an $\mathfrak{R}$-linear map. If F_1 has a basis $(v_1, \ldots, v_n)$ and F_2 has a basis $(w_1, \ldots, w_m)$, the matrix $(a_{ij})_{i=1,\ldots,n, j=1,\ldots,m}$ where a_{ij}s satisfy $f(v_i) = a_{i1} w_1 + \cdots + a_{im} w_m$ for any $i = 1, \ldots, n$ is called a *matrix representation* of f.

We define submodules and quotient modules, as in linear algebra.

▶ **Definition 14.** *Let $(M, +, \cdot)$ be a left (resp. right) $\mathfrak{R}$-module. A subgroup N of $(M, +)$ is a* submodule *if for any $x \in N$ and $r \in \mathfrak{R}$, the scalar multiplication $r \cdot x$ (resp. $x \cdot r$) is in N.*

For any submodule N, the quotient group M/N is also an $\mathfrak{R}$-module. M/N is called the quotient module *of M by N.*

For submodules and quotient modules, the following basic theorems are known:

▶ **Theorem 15** (First isomorphism theorem). *[7, Theorem 7.8] Let $(M, +, \cdot), (M', +', \cdot')$ be left (or right) $\mathfrak{R}$-modules, and $f : M \to M'$ be an $\mathfrak{R}$-linear map.*

1. *The inverse image of* 0 *by* f, $\ker f = \{x \in M \mid f(x) = 0\}$, *is a submodule of* M.
2. *The image of* M *by* f, $\operatorname{im} f = \{f(x) \mid x \in M\}$, *is a submodule of* M'.
3. *The image* $\operatorname{im} f$ *is isomorphic to* $M/\ker f$.

▶ **Theorem 16** (Third isomorphism theorem). *[7, Theorem 7.10] Let M be a left (or right) $\mathfrak{R}$-module, N be a submodule of M, and L be a submodule of N. Then $(M/L)/(N/L)$ is isomorphic to M/N.*

▶ **Theorem 17.** *[7, Theorem 9.8] Let $\mathfrak{R}$ be $\mathbb{Z}$ or $\mathbb{Z}/p\mathbb{Z}$ for some prime p. Every submodule of a free $\mathfrak{R}$-module is free. Moreover, if an $\mathfrak{R}$-module M is isomorphic to $\mathfrak{R}^n$, then every submodule N of M is isomorphic to $\mathfrak{R}^m$ for some $m \leq n$. (In general, this holds for any principal ideal domain $\mathfrak{R}$.)*

Let M be a left $\mathfrak{R}$-module. For $S \subset M$, the set $\mathfrak{R}S$ of all elements in M of the form $\sum_{i=1}^{k} r_i s_i$ ($k \in \mathbb{Z}_{\geq 0}, r_i \in \mathfrak{R}, s_i \in S$) is a submodule of M. If $\mathfrak{R}S = M$, S is called a *generating set* of S and the elements of S are called *generators* of M. Let $S = \{s_i\}_{i \in I}$ be a generating set of M for some indexing set I. For a set $X = \{x_i\}_{i \in I}$, the linear map $\epsilon : \mathfrak{R}\underline{X} \ni x_i \mapsto s_i \in M$ is a surjection from the free module $\mathfrak{R}\underline{X}$. The elements of $\ker \epsilon$, that is, elements $\sum_{x_i \in X} r_i \underline{x_i}$ satisfying $\epsilon(\sum_{x_i \in X} r_i \underline{x_i}) = \sum_{x_i \in X} r_i s_i = 0$, are called *relations* of M.

Now, we introduce one of the most important notions to develop the homology theory of rewriting systems, *free resolutions*. We first start from the following example.

▶ **Example 18.** Let M be the $\mathbb{Z}$-module defined by

$$\mathbb{Z}\underline{\{a, b, c, d, e\}}/\mathbb{Z}\{\underline{a} + \underline{b} + \underline{c} - \underline{d} - \underline{e},\ 2\underline{b} - \underline{c},\ \underline{a} + 2\underline{c} - \underline{b} - \underline{d} - \underline{e}\}.$$

We consider the $\mathbb{Z}$-linear map between free $\mathbb{Z}$-modules $f_0 : \mathbb{Z}^3 \to \mathbb{Z}\underline{\{a, b, c, d, e\}}$ defined by

$$f_0(1, 0, 0) = \underline{a} + \underline{b} + \underline{c} - \underline{d} - \underline{e},\ f_0(0, 1, 0) = 2\underline{b} - \underline{c},\ f_0(0, 0, 1) = \underline{a} + 2\underline{c} - \underline{b} - \underline{d} - \underline{e}.$$

We can see that the image of f_0 is the set of relations of M. In other words, $\operatorname{im} f_0 = \ker \epsilon$ for the linear map $\epsilon : \mathbb{Z}\underline{\{a, b, c, d, e\}} \to M$ which maps each element to its equivalence class. Then, we consider the "relations between relations", that is, triples (n_1, n_2, n_3) which satisfy $f_0(n_1, n_2, n_3) = n_1(\underline{a} + \underline{b} + \underline{c} - \underline{d} - \underline{e}) + n_2(2\underline{b} - \underline{c}) + n_3(\underline{a} + 2\underline{c} - \underline{b} - \underline{d} - \underline{e}) = 0$, or equivalently, elements of $\ker f_0$. We can check $\ker f_0 = \{m(-1, 1, 1) \mid m \in \mathbb{Z}\}$. This fact can be explained in terms of rewriting systems. If we write relations in the form of rewrite rules

$$A_1.\ \underline{a} + \underline{b} + \underline{c} \to \underline{d} + \underline{e},\ A_2.\ 2\underline{b} \to \underline{c},\ A_3.\ \underline{a} + 2\underline{c} \to \underline{b} + \underline{d} + \underline{e},$$

we see $\{A_1, A_2, A_3\}$ is a complete rewriting system with two joinable critical pairs

$$2\underline{b} + \underline{d} + \underline{e} \xleftarrow{A_3} \underline{a} + \underline{b} + 2\underline{c} \xrightarrow{A_1} \underline{c} + \underline{d} + \underline{e}, \quad 2\underline{b} + \underline{d} + \underline{e} \xrightarrow{A_2} \underline{c} + \underline{d} + \underline{e}$$

$$\underline{a} + 2\underline{c} \xleftarrow{A_2} \underline{a} + 2\underline{b} + \underline{c} \xrightarrow{A_1} \underline{b} + \underline{d} + \underline{e}, \quad \underline{a} + 2\underline{c} \xrightarrow{A_3} \underline{b} + \underline{d} + \underline{e}.$$

We associate these critical pairs with an equality between formal sums $A_2 + A_3 = A_1$, and it corresponds to

$$f_0(-1, 1, 1) = \underbrace{-(\underline{a} + \underline{b} + \underline{c} - \underline{d} - \underline{e})}_{-A_1} + \underbrace{(2\underline{b} - \underline{c})}_{A_2} + \underbrace{(\underline{a} + 2\underline{c} - \underline{b} - \underline{d} - \underline{e})}_{A_3} = 0.$$

In fact, this correspondence between critical pairs and "relations between relations" is a key to the homology theory of TRSs.

We define a linear map $f_1 : \mathbb{Z} \to \mathbb{Z}^3$ by $f_1(1) = (-1, 1, 1)$ and then f_1 satisfies $\operatorname{im} f_1 = \ker f_0$. We can go further, that is, we can consider $\ker f_1$, but it clearly turns out that $\ker f_1 = 0$.

We encode the above information in the following diagram:

$$\mathbb{Z} \xrightarrow{f_1} \mathbb{Z}^3 \xrightarrow{f_0} \mathbb{Z}\underline{\{a, b, c, d, e\}} \xrightarrow{\epsilon} M \tag{3}$$

where $\operatorname{im} f_1 = \ker f_0, \operatorname{im} f_0 = \ker \epsilon$ and ϵ is surjective. Sequences of modules and linear maps with these conditions are called free resolutions:

▶ **Definition 19.** *A sequence of left $\mathfrak{R}$-modules and $\mathfrak{R}$-linear maps*

$$\cdots \xrightarrow{f_{i+1}} M_{i+1} \xrightarrow{f_i} M_i \xrightarrow{f_{i-1}} \cdots$$

is called an exact sequence *if* $\operatorname{im} f_i = \ker f_{i-1}$ *holds for any* i.

Let M be a left $\mathfrak{R}$-module. For infinite sequence of free modules F_i and linear maps $f_i : F_{i+1} \to F_i$, $\epsilon : F_0 \to M$, if the sequence

$$\cdots \xrightarrow{f_1} F_1 \xrightarrow{f_0} F_0 \xrightarrow{\epsilon} M$$

is exact and ϵ is surjective, the sequence above is called a free resolution *of M. If the sequence is finite, it is called a* partial free resolution.

(Exact sequences and free resolutions are defined for right $\mathfrak{R}$-modules in the same way.)

Notice that the exact sequence (3) can be extended to the infinite exact sequence

$$\cdots \to 0 \to \cdots \to 0 \to \mathbb{Z} \xrightarrow{f_1} \mathbb{Z}^3 \xrightarrow{f_0} \mathbb{Z}\underline{\{a, b, c, d, e\}} \xrightarrow{\epsilon} M$$

since $\ker f_1 = 0$. Thus, the sequence (3) is a free resolution of M.

As there are generally several rewriting systems equivalent to a given equational theory, free resolutions of M are not unique. However, we can construct some information of M from a (partial) free resolution which does not depend on the choice of the free resolution. The information is called *homology groups*. To define the homology groups, we introduce the tensor product of modules.

▶ **Definition 20.** *Let N be a right $\mathfrak{R}$-module and M be a left $\mathfrak{R}$-module. Let $F(N \times M)$ be the free abelian group generated by $N \times M$. The* tensor product *of N and M, denoted by $N \otimes_{\mathfrak{R}} M$, is the quotient group of $F(N \times M)$ by the subgroup generated by the elements of the form*

$$(x, y) + (x, y') - (x, y + y'),\ (x, y) + (x', y) - (x + x', y),\ (x \cdot r, y) - (x, r \cdot y)$$

where $x, x' \in N$, $y, y' \in M$, $r \in R$. The equivalence class of (x, y) in $N \otimes_{\mathfrak{R}} M$ is written as $x \otimes y$.

For a right $\mathfrak{R}$-module N and a $\mathfrak{R}$-linear map $f : M \to M'$ between left $\mathfrak{R}$-modules M, M', we write $N \otimes f : N \otimes_{\mathfrak{R}} M \to N \otimes_{\mathfrak{R}} M'$ for the map $(N \otimes f)(a \otimes x) = a \otimes f(x)$. $N \otimes f$ is known to be well-defined and be a group homomorphism.

Let $\cdots \xrightarrow{f_1} F_1 \xrightarrow{f_0} F_0 \xrightarrow{\epsilon} M$ be a free resolution of a left $\mathfrak{R}$-module M. For a right $\mathfrak{R}$-module N, we consider the sequence

$$\cdots \xrightarrow{N \otimes f_1} N \otimes_{\mathfrak{R}} F_1 \xrightarrow{N \otimes f_0} N \otimes_{\mathfrak{R}} F_0. \tag{4}$$

Then, it can be shown that $\mathrm{im}(N \otimes f_i) \subset \ker(N \otimes f_{i-1})$ for any $i = 1, 2, \ldots$. In general, a sequence $\cdots \xrightarrow{f_{i+1}} M_{i+1} \xrightarrow{f_i} M_i \xrightarrow{f_{i-1}} \cdots$ of left/right $\mathfrak{R}$-modules satisfying $\mathrm{im}\, f_i \subset \ker f_{i-1}$ for any i is called a *chain complex*. The homology groups of a chain complex are defined to be the quotient group of $\ker f_{i-1}$ by $\mathrm{im}\, f_i$:

▶ **Definition 21.** *Let* $(C_\bullet, f_\bullet)$ *denote the pair* $(\{C_i\}_{i=0,1,\ldots}, \{f_i : C_{i+1} \to C_i\}_{i=0,1,\ldots})$. *For a chain complex* $\cdots \xrightarrow{f_{i+1}} C_{i+1} \xrightarrow{f_i} C_i \xrightarrow{f_{i-1}} \cdots$, *the abelian group* $H_j(C_\bullet, f_\bullet)$ *defined by*

$$H_j(C_\bullet, f_\bullet) = \ker f_{j-1} / \mathrm{im}\, f_j$$

is called the j-th homology groups *of the chain complex* $(C_\bullet, f_\bullet)$.

The homology groups of the chain complex (4) depend only on M, N, and $\mathfrak{R}$:

▶ **Theorem 22.** *[6, Corollary 6.21] Let* M *be a left* $\mathfrak{R}$*-module and* N *be a right* $\mathfrak{R}$*-module. For any two resolutions* $\cdots \xrightarrow{f_1} F_1 \xrightarrow{f_0} F_0 \xrightarrow{\epsilon} M$, $\cdots \xrightarrow{f'_1} F'_1 \xrightarrow{f'_0} F'_0 \xrightarrow{\epsilon} M$, *we have a group isomorphism*

$$H_j(N \otimes_{\mathfrak{R}} F_\bullet, N \otimes f_\bullet) \cong H_j(N \otimes_{\mathfrak{R}} F'_\bullet, N \otimes f'_\bullet).$$

We end this subsection by giving some basic facts on exact sequences.

▶ **Proposition 23.** *[7, Proposition 7.20 and 7.21]*

1. $M_1 \xrightarrow{f} M_2 \to 0$ *is exact if and only if* $\ker f = 0$.
2. $0 \to M_1 \xrightarrow{f} M_2$ *is exact if and only if* $\mathrm{im}\, f = M_2$.
3. *If* M_1 *is a submodule of* M_2*, the sequence* $0 \to M_2 \xrightarrow{\iota} M_1 \xrightarrow{\pi} M_1/M_2 \to 0$ *is exact where* ι *is the inclusion map* $\iota(x) = x$ *and* π *is the projection* $\pi(x) = [x]$.

▶ **Proposition 24.** *Suppose we have an exact sequence of* $\mathfrak{R}$*-modules* $0 \to M_1 \to M_2 \to M_3 \to 0$. *If* M_3 *is free, then* M_2 *is isomorphic to* $M_1 \times M_3$.
The proof is given by using [7, Proposition 7.22].

3.2 String Rewriting Systems and Homology Groups of Monoids

For an alphabet Σ, Σ^* denotes the set of all strings of symbols over Σ. Σ^* forms a monoid under the operation of concatenation with the empty string serving as the identity, and we call Σ^* the free monoid generated by Σ. For a string rewriting system (SRS) (Σ, R), we write $\mathcal{M}_{(\Sigma,R)}$ for the set defined by $\mathcal{M}_{(\Sigma,R)} = \Sigma^* / \xleftrightarrow{*}_R$. We can see $\mathcal{M}_{(\Sigma,R)}$ is a monoid under the operations $[u] \cdot [v] = [uv]$ where $[w]$ denotes the equivalence class of $w \in \Sigma^*$ with respect to $\xleftrightarrow{*}_R$.

We say that two SRSs $(\Sigma_1, R_1), (\Sigma_2, R_2)$ are *isomorphic* if the monoids $\mathcal{M}_{(\Sigma_1,R_1)}, \mathcal{M}_{(\Sigma_2,R_2)}$ are isomorphic. It is not difficult to show that for any two SRSs $(\Sigma, R_1), (\Sigma, R_2)$ with the same signature, if R_1 and R_2 are equivalent (i.e., $\xleftrightarrow{*}_{R_1} = \xleftrightarrow{*}_{R_2}$), then (Σ, R_1) and (Σ, R_2) are isomorphic. Roughly speaking, the notion that two SRSs are isomorphic means that the SRSs are equivalent but their alphabets can be different. For example, let Σ_1 be $\{a, b, c\}$ and R_1 be $\{abb \to ab,\ ba \to c\}$. Then, (Σ_1, R_1) is isomorphic to (Σ_2, R_2) where $\Sigma_2 = \{a, b\}$ and $R_2 = \{abb \to ab\}$. Intuitively, since c is equivalent to ba with respect to the congruence $\xleftrightarrow{*}_{R_1}$, c is redundant as long as we consider strings modulo $\xleftrightarrow{*}_{R_1}$ and (Σ_2, R_2) is the SRS made by removing c from (Σ_1, R_1).

If a monoid S is isomorphic to $\mathcal{M}_{(\Sigma,R)}$ for an SRS (Σ, R), we call (Σ, R) a *presentation* of the monoid S.

Let S be a monoid and consider the free $\mathbb{Z}$-module $\mathbb{Z}\underline{S}$. $\mathbb{Z}\underline{S}$ can be equipped with a ring structure under the multiplication $\left(\sum_{w\in S} n_w \underline{w}\right)\left(\sum_{w\in S} m_w \underline{w}\right) = \sum_{w,v\in S} n_w m_v \underline{wv}$ where $n_w m_v$ is the usual multiplication of integers and wv is the multiplication of the monoid S. $\mathbb{Z}\underline{S}$ as a ring is called the *integral monoid ring* of S. When we think of $\mathbb{Z}\underline{S}$ as a ring, we write $\mathbb{Z}\langle S\rangle$ instead of $\mathbb{Z}\underline{S}$.

We consider $\mathbb{Z}\langle S\rangle$-modules. The group of integers $\mathbb{Z}$ forms a left (resp. right) $\mathbb{Z}\langle S\rangle$-module under the scalar multiplication $(\sum_{w\in S} n_w \underline{w}) \cdot m = \sum_{w\in S} n_w m \underline{w}$ (resp. $m \cdot (\sum_{w\in S} n_w \underline{w}) = \sum_{w\in S} n_w m \underline{w}$). Let $\cdots \xrightarrow{\partial_1} F_1 \xrightarrow{\partial_0} F_0 \xrightarrow{\epsilon} \mathbb{Z}$ be a free resolution of $\mathbb{Z}$ over the ring $\mathbb{Z}\langle S\rangle$. The *$i$-th monoid homology* $H_i(S)$ is defined as the i-th homology group of the chain complex $(\mathbb{Z} \otimes_{\mathbb{Z}\langle S\rangle} F_\bullet, \mathbb{Z} \otimes \partial_\bullet)$, i.e.,

$$H_i(S) = H_i(\mathbb{Z} \otimes_{\mathbb{Z}\langle S\rangle} F_\bullet, \mathbb{Z} \otimes \partial_\bullet) = \ker \mathbb{Z} \otimes \partial_{i-1} / \operatorname{im} \mathbb{Z} \otimes \partial_i.$$

If S is isomorphic to $\mathcal{M}_{(\Sigma,R)}$ for some SRS (Σ, R), it is known that there is a free resolution in the form of

$$\cdots \to (\mathbb{Z}\langle S\rangle)\underline{P} \xrightarrow{\partial_2} (\mathbb{Z}\langle S\rangle)\underline{R} \xrightarrow{\partial_1} (\mathbb{Z}\langle S\rangle)\underline{\Sigma} \xrightarrow{\partial_0} (\mathbb{Z}\langle S\rangle)\underline{\{\star\}} \xrightarrow{\epsilon} \mathbb{Z}$$

for some set P. Squier [8] showed that if the SRS (Σ, R) is complete and reduced[1], there is $\partial_2 : (\mathbb{Z}\langle S\rangle)\underline{P} \to (\mathbb{Z}\langle S\rangle)\underline{R}$ for $P =$ (the critical pairs of R) so that we can compute $H_2(S) = \ker \partial_1 / \operatorname{im} \partial_2$ explicitly. This is an analog of Example 18, but we omit the details here. For an abelian group G, let $s(G)$ denote the minimum number of generators of G (i.e., the minimum cardinality of the subset $A \subset G$ such that any element $x \in G$ can be written by $x = a_1 + \cdots + a_k - a_{k+1} - \cdots - a_m$ for $a_1, \ldots, a_m \in A$). Then, we have the following theorem:

▶ **Theorem 25.** *Let (Σ, R) be an SRS and $S = \mathcal{M}_{(\Sigma,R)}$. Then $\#\Sigma \geq s(H_1(S))$, $\#R \geq s(H_2(S))$.*

To prove this theorem, we use the following lemma:

▶ **Lemma 26.** *Let X be a set. The group homomorphism $\mathbb{Z} \otimes_{\mathbb{Z}\langle S\rangle} (\mathbb{Z}\langle S\rangle)\underline{X} \to \mathbb{Z}\underline{X}$, $n\langle w\rangle\underline{x} \mapsto n\underline{x}$ is an isomorphism.*

This lemma is proved in a straightforward way.

Proof of Theorem 25. Since $\mathbb{Z} \otimes_{\mathbb{Z}\langle S\rangle} (\mathbb{Z}\langle S\rangle)\underline{X} \cong \mathbb{Z}\underline{X}$ by the above lemma, $s(\mathbb{Z} \otimes_{\mathbb{Z}\langle S\rangle} (\mathbb{Z}\langle S\rangle)\underline{X}) = s(\mathbb{Z}\underline{X}) = \#X$. For any set Y and group homomorphism $f : \mathbb{Z}\underline{X} \to \mathbb{Z}\underline{Y}$, since $\ker f$ is a subgroup of $\mathbb{Z}\underline{X}$, we have $\#X \geq s(\ker f)$. For any subgroup H of $\ker f$, $\ker f / H$ is generated by $[x_1], \ldots, [x_k]$ if $\ker f$ is generated by $x_1, \ldots, x_k$. Thus $\#\Sigma \geq s(\ker \partial_0 / \operatorname{im} \partial_1) = s(H_1(S))$, $\#R \geq s(\ker \partial_1 / \operatorname{im} \partial_2) = s(H_2(S))$. ◀

Note that $H_i(S)$ does not depend on the choice of presentation (Σ, R) by Theorem 22. Therefore, Theorem 25 can be restated as follows: Let (Σ, R) be an SRS. For any SRS (Σ', R') isomorphic to (Σ, R), the number of symbols $\#\Sigma'$ is bounded below by $s(H_1(\mathcal{M}_{(\Sigma,R)}))$ and the number of rules $\#R'$ is bounded below by $s(H_2(\mathcal{M}_{(\Sigma,R)}))$.

[1] An SRS (Σ, R) is reduced if for each $l \to r \in R$, r is normal w.r.t. $\to_R$ and there does not exist $l' \to r' \in R$ such that $l' = ulv \neq l$ for some $u, v \in \Sigma^*$

4 An Overview of the Homology Theory of TRSs

In this section, we will briefly see the homology theory of TRSs, which is the main tool to obtain our lower bounds.

We fix a signature Σ. Let $t = \langle t_1, \dots, t_n \rangle$ be a n-uple of terms and suppose that for each t_i, the set of variables in t_i is included in $\{x_1, \dots, x_m\}$. For an m-uple of term $s = \langle s_1, \dots, s_m \rangle$, we define the composition of t and s by

$$t \circ s = \langle t_1[s_1/x_1, \dots, s_m/x_m], \dots, t_n[s_1/x_1, \dots, s_m/x_m] \rangle$$

where $t_i[s_1/x_1, \dots, s_m/x_m]$ denotes the term obtained by substituting s_j for x_j in t_i for each $j = 1, \dots, m$ in parallel. (For example, $f(x_1, x_2)[g(x_2)/x_1, g(x_1)/x_2] = f(g(x_2), g(x_1))$.) By this definition, we can think of any m-uple $\langle s_1, \dots, s_m \rangle$ of terms as a (parallel) substitution $\{x_1 \mapsto s_1, \dots, x_m \mapsto s_m\}$. Recall that, for a TRS R, the reduction relation $\to_R$ between terms is defined as $t_1 \to_R t_2 \iff t_1 = C[l \circ s],\ t_2 = C[r \circ s]$ for some single-hole context C, m-uple s of terms, and rewrite rule $l \to r \in R$ whose variables are included in $\{x_1, \dots, x_m\}$. This definition suggests that the pair of a context C and an m-uple of terms (or equivalently, substitution) s is useful to think about rewrite relations. Malbos and Mimram [3] called the pair of a context and an m-uple of terms a *bicontext*. For a bicontext (C, t) and a rewrite rule A, we call the triple (C, A, t) a *rewriting step*. The pair of two rewriting steps $(\square, l_1 \to r_1, s), (C, l_2 \to r_2, t)$ is called a *critical pair* if the pair $(r_1 \circ s, C[r_2 \circ t])$ of terms is a critical pair in the usual sense given by $l_1 \to r_1$, $l_2 \to r_2$.

The composition of two bicontexts $(C, t), (D, s)$ $(t = \langle t_1, \dots, t_n \rangle,\ s = \langle s_1, \dots, s_m \rangle)$ is defined by

$$(C, t) \circ (D, s) = (C[D \circ t], s \circ t)$$

where $D \circ t = D[t_1/x_1, \dots, t_n/x_n]$ and note that the order of composition is reversed in the second component. With this composition, we can define the small category of bicontexts $\mathbb{K}$ as

- Objects : natural numbers,
- Morphisms $\mathbb{K}(n, m)$ $(n, m \in \mathbb{N})$: bicontexts (C, t) where $t = \langle t_1, \dots, t_n \rangle$ and each t_i and C have variables in $\{x_1, \dots, x_m\}$ (except $\square$ in C),
- Identity $\mathrm{id}_n = (\square, \langle x_1, \dots, x_n \rangle)$,
- Composition $\circ : \mathbb{K}(n, m) \times \mathbb{K}(k, n) \to \mathbb{K}(k, m)$: defined above.

To apply homological algebra to TRSs, we construct an algebraic structure from $\mathbb{K}$. We write $\mathbb{Z}\langle\mathbb{K}\rangle$ for the (small) category whose objects are natural numbers, set of morphisms $(\mathbb{Z}\langle\mathbb{K}\rangle)(n, m)$ is the free abelian group generated by $\mathbb{K}(n, m)$ (i.e., any element in $(\mathbb{Z}\langle\mathbb{K}\rangle)(n, m)$ is written in the form of formal sum $\sum_{(C,t) \in \mathbb{K}(n,m)} \lambda_{(C,t)}(C, t)$ where each $\lambda_{(C,t)}$ is in $\mathbb{Z}$ and is equal to 0 except for finitely many (C, t)s). The composition on $\mathbb{Z}\langle\mathbb{K}\rangle$ is defined by

$$\left(\sum_{(C,t)} \lambda_{(C,t)}(C, t)\right) \circ \left(\sum_{(D,s)} \mu_{(D,s)}(D, s)\right) = \sum_{(C,t)} \sum_{(D,s)} \lambda_{(C,t)}\mu_{(D,s)}((C, t) \circ (D, s)).$$

By this definition, we can see that this composition $\circ$ is bilinear, that is,

$$a \circ (b_1 + b_2) = a \circ b_1 + a \circ b_2, \tag{5}$$

$$(a_1 + a_1) \circ b = a_1 \circ b + a_2 \circ b \tag{6}$$

for any $a, a_1, a_2 \in (\mathbb{Z}\langle\mathbb{K}\rangle)(n,m)$, $b, b_1, b_2 \in (\mathbb{Z}\langle\mathbb{K}\rangle)(k,n)$. Also, we have

$$a \circ 0 = 0 \circ b = 0. \tag{7}$$

One may notice that this looks something similar to the ring structure. Indeed, $\mathbb{Z}\langle\mathbb{K}\rangle$ forms the structure called *ringoid*, which is defined as follows:

▶ **Definition 27.** *A ringoid $\mathcal{R}$ is a small category in which each hom-set $\mathcal{R}(X,Y)$ is equipped with a structure of abelian group $(\mathcal{R}(X,Y),+)$ and satisfies (5, 6, 7).*

Intuitively, a ringoid $\mathcal{R}$ is a "multi-sorted" ring where sorts are the hom-sets $\mathcal{R}(X,Y)$ for any objects X, Y of $\mathcal{R}$ and it has an addition $+ : \mathcal{R}(X,Y) \times \mathcal{R}(X,Y) \to \mathcal{R}(X,Y)$ for each pair (X,Y) of objects, and a multiplication $\circ : \mathcal{R}(Y,Z) \times \mathcal{R}(X,Y) \to \mathcal{R}(X,Z)$ for each triple (X,Y,Z) of objects. If $\mathcal{R}$ has exactly one object $\star$, the ringoid $\mathcal{R}$ can be identified with the ring $(\mathcal{R}(\star,\star),+,\circ)$. (Note that the morphisms correspond to the elements of the ring, not objects.) We can also define modules over a ringoid. For a ringoid $\mathcal{R}$, a left $\mathcal{R}$-module M associates each object X of $\mathcal{R}$ with an abelian group $M(X)$ and has a "multi-sorted" scalar multiplication $\cdot : \mathcal{R}(X,Y) \times M(X) \to M(Y)$ for each pair of objects X, Y of $\mathcal{R}$. This notion is interpreted as a functor from the category $\mathcal{R}$ to the category of abelian groups:

▶ **Definition 28.** *Let $\mathcal{R}$ be a ringoid. A left $\mathcal{R}$-module is a functor $M : \mathcal{R} \to \mathbf{Ab}$ satisfying*

$$M(a+b) = M(a) + M(b), \ \ M(0) = 0 \quad (a, b \in \mathcal{R}(X,Y), X, Y \in \mathrm{Obj}(\mathcal{R}))$$

where $\mathbf{Ab}$ is the category of abelian groups. We define the scalar multiplication $\cdot : \mathcal{R}(X,Y) \times M(X) \to M(Y)$ by $a \cdot m = M(a)(m)$.

A right $\mathcal{R}$-module is defined as a left $\mathcal{R}^{\mathrm{op}}$ module.

For two left $\mathcal{R}$-modules M_1, M_2, an $\mathcal{R}$-linear map $f : M_1 \to M_2$ is a natural transformation such that each component $f_X : M_1(X) \to M_2(X)$ is a group homomorphism.

If $\mathcal{R}$ has exactly one object $\star$, M can be identified with the left $\mathcal{R}(\star,\star)$-module $(M(\star),+,\cdot)$. A free $\mathcal{R}$-module is defined as follows.

▶ **Definition 29.** *Let $\mathcal{R}$ be a ringoid and P be a family of sets P_X $(X \in \mathrm{Obj}(\mathcal{R}))$. The free left $\mathcal{R}$-module generated by P, denoted by $\mathcal{R}\underline{P}$ is defined as follows. For each object $X \in \mathrm{Obj}(\mathcal{R})$, $(\mathcal{R}\underline{P})(X)$ is the abelian group of formal finite sums*

$$\sum_{x_Y \in P_Y,\ Y \in \mathrm{Obj}(\mathcal{R})} a_{x_Y} \underline{x_Y}, \quad (a_{x_Y} \in \mathcal{R}(Y,X))$$

and for each morphism $r \in \mathcal{R}(X,Z)$,

$$r \cdot \left(\sum_{x_Y \in P_Y,\ Y \in \mathrm{Obj}(\mathcal{R})} a_{x_Y} \underline{x_Y} \right) = \sum_{x_Y \in P_Y,\ Y \in \mathrm{Obj}(\mathcal{R})} (r \circ a_{x_Y}) \underline{x}_Y.$$

For $\mathbb{Z}\langle\mathbb{K}\rangle$, we write $C\underline{x}t$ for elements of $((\mathbb{Z}\langle\mathbb{K}\rangle)\underline{P})(X)$ instead of $(C,t)\underline{x}$, and $(D+C)\underline{x}t$ for $D\underline{x}t + C\underline{x}t$.

The tensor product of two modules over a ringoid is also defined.

▶ **Definition 30.** *Let $\mathcal{R}$ be a ringoid, M_1 be a right $\mathcal{R}$-module, and M_2 be a left $\mathcal{R}$-module. For a family of groups $\{G_X \mid X \in P\}$ for some indexing set P, its direct sum, denoted by $\bigoplus_{X \in P} G_X$, is the subset of the direct product defined by $\{(g_X)_{X \in P} \in \prod_{X \in P} G_X \mid g_X = 0$ except for finite Xs$\}$. The direct sum of groups also forms a group.*

The tensor product $M_1 \otimes_{\mathcal{R}} M_2$ is the quotient abelian group of $\bigoplus_{X \in \mathcal{R}} M_1(X) \otimes_{\mathcal{R}(X,X)} M_2(X)$ by relations $(a^{\mathrm{op}} \cdot x) \otimes y - x \otimes (a \cdot y)$ for all $a \in \mathcal{R}(Y,X)$, $x \in M(X), y \in M(Y)$.

Now, we outline Malbos-Mimram's construction of the homology groups of TRSs.

1. We begin by defining the quotient ringoid $\overline{\mathbb{Z}\langle\mathbb{K}\rangle}^{(\Sigma,R)}$ of $\mathbb{Z}\langle\mathbb{K}\rangle$ by some relations so that $\overline{\mathbb{Z}\langle\mathbb{K}\rangle}^{(\Sigma,R)}$ depends only on the Tietze equivalence class of (Σ, R). $\overline{\mathbb{Z}\langle\mathbb{K}\rangle}^{(\Sigma,R)}$ corresponds to $\mathcal{M}_{(\Sigma,R)}$ in the case (Σ, R) is an SRS.
2. From this step, we write $\mathcal{R}$ for $\overline{\mathbb{Z}\langle\mathbb{K}\rangle}^{(\Sigma,R)}$. It can be shown that we have a partial free resolution

$$\mathcal{R}\underline{\mathbf{P}_3} \xrightarrow{\partial_2} \mathcal{R}\underline{\mathbf{P}_2} \xrightarrow{\partial_1} \mathcal{R}\underline{\mathbf{P}_1} \xrightarrow{\partial_0} \mathcal{R}\underline{\mathbf{P}_0} \xrightarrow{\epsilon} \mathcal{Z}$$

where every $\mathbf{P}_i$ is a family of sets $(\mathbf{P}_i)_j$ given by $(\mathbf{P}_0)_1 = \{1\}$, $(\mathbf{P}_0)_j = \emptyset$ $(j \neq 1)$, $(\mathbf{P}_1)_j = \Sigma^{(j)} = \{f \in \Sigma \mid f \text{ is of arity } j\}$, $(\mathbf{P}_2)_j = \{l \to r \in R \mid l \to r \text{ is of arity } j\}$, $(\mathbf{P}_3)_j = \{((\Box, A, s), (C, B, t)) : \text{critical pair} \mid A, B \in (\mathbf{P}_2)_j\}$. $\mathcal{Z}$ is a left $\mathcal{R}$-module defined as the quotient of $\mathcal{R}\underline{P}$ by all relations of the form $\sum_i (\kappa_i(u) \circ t)\underline{\star}\langle t_i\rangle - \Box\underline{\star}\langle u \circ t\rangle$ for every term $u \circ t$ where $P_1 = \{\star\}$, $P_j = \emptyset$ $(j \neq 1)$ and κ_i is defined later.
3. By taking the tensor product $\mathbb{Z}\otimes_{\mathcal{R}}$, we have the chain complex

$$\mathbb{Z} \otimes_{\mathcal{R}} \mathcal{R}\underline{\mathbf{P}_3} \xrightarrow{\mathbb{Z}\otimes\partial_2} \mathbb{Z} \otimes_{\mathcal{R}} \mathcal{R}\underline{\mathbf{P}_2} \xrightarrow{\mathbb{Z}\otimes\partial_1} \mathbb{Z} \otimes_{\mathcal{R}} \mathcal{R}\underline{\mathbf{P}_1} \xrightarrow{\mathbb{Z}\otimes\partial_0} \mathbb{Z} \otimes_{\mathcal{R}} \mathcal{R}\underline{\mathbf{P}_0} \tag{8}$$

where $\mathbb{Z}$ above is the $\mathcal{R}$-module defined by $\mathbb{Z}(i) = \mathbb{Z}$ (the abelian group of integers) for each object i, and the scalar multiplication is given by $(C, t) \cdot k = k$.
4. The homology groups can be defined by

$$H_i(\Sigma, R) = \ker(\mathbb{Z} \otimes \partial_{i-1}) / \operatorname{im}(\mathbb{Z} \otimes \partial_i).$$

It is shown that the homology groups of TRS depend only on the "Tietze equivalence" class of (Σ, R). Tietze equivalence is an analog of isomorphism between SRSs; it is an equivalence between two TRSs (Σ_1, R_1), (Σ_2, R_2) where the signatures Σ_1 and Σ_2 can be different, while the usual equivalence is defined for TRSs with the same signature by $\overset{*}{\leftrightarrow}_{R_1} = \overset{*}{\leftrightarrow}_{R_2}$. Especially, any two TRSs (Σ, R_1),(Σ, R_2) are Tietze equivalent if they are equivalent in the usual sense, $\overset{*}{\leftrightarrow}_{R_1} = \overset{*}{\leftrightarrow}_{R_2}$. Thus, we have the following:

$$\overset{*}{\leftrightarrow}_{R_1} = \overset{*}{\leftrightarrow}_{R_2} \implies H_i(\Sigma, R_1) \cong H_i(\Sigma, R_2).$$

For the step 1, we define the relations of $\overline{\mathbb{Z}\langle\mathbb{K}\rangle}^{(\Sigma,R)}$. We identify elements in $\mathbb{Z}\langle\mathbb{K}\rangle$ as follows. (a) For two m-uples $t = \langle t_1, \dots, t_m\rangle, s = \langle s_1, \dots, s_m\rangle$ of terms, we identify t and s if $t \overset{*}{\leftrightarrow}_R s$. (b) Similarly, for two single-hole contexts C, D, we identify C and D if $C \overset{*}{\leftrightarrow}_R D$. For the last identification, we introduce operator κ_i which takes a term t and returns the formal sum of single-hole contexts $C_1 + \cdots + C_m$ where C_j $(j = 1, \dots, m)$ is obtained by replacing the j-th occurrence of x_i with $\Box$ in t, and m is the number of the occurrences of x_i in t. For example, we have

$$\begin{aligned}
\kappa_1(f(g(x_1, x_2), x_1)) &= f(g(\Box, x_2), x_1) + f(g(x_1, x_2), \Box),\\
\kappa_2(f(g(x_1, x_2), x_1)) &= f(g(x_1, \Box), x_1),\\
\kappa_2(h(x_1)) &= 0.
\end{aligned}$$

The definition of κ_i can be stated inductively as follows:

$$\begin{aligned}
\kappa_i(x_i) = \Box,\ \kappa_i(x_j) &= 0 \quad (j \neq i),\\
\kappa_i(f(t_1, \dots, t_n)) &= \sum_{k=1}^{n} f(t_1, \dots, t_{k-1}, \kappa_i(t_k), t_{k+1}, \dots, t_n).
\end{aligned}$$

Then, (c) we identify formal sums of bicontexts $(C_1, t) + \cdots + (C_k, t)$ and $(D_1, t) + \cdots + (D_l, t)$ if $\kappa_i(u) = C_1 + \cdots + C_k$, $\kappa_i(v) = D_1 + \cdots + D_l$ for some positive integer i and terms u, v such that $u \xleftrightarrow{*}_R v$. $\overline{\mathbb{Z}\langle\mathbb{K}\rangle}^{(\Sigma,R)}$ is defined as the quotient of $\mathbb{Z}\langle\mathbb{K}\rangle$ by the equivalence class generated by the identifications (a), (b), and (c).

We omit the definitions of the $\mathcal{R}$-linear maps ϵ, ∂_i $(i = 0, 1, 2)$ in the step 2, but we describe the group homomorphisms $\mathbb{Z} \otimes \partial_i : \mathbb{Z} \otimes_{\mathcal{R}} \mathcal{R}\underline{\mathbf{P}_{i+1}} \to \mathbb{Z} \otimes_{\mathcal{R}} \mathcal{R}\underline{\mathbf{P}_i}$. Let $\tilde{\partial}_i$ denote $\mathbb{Z} \otimes \partial_i$ for simplicity. For the step 2, we define the $\mathcal{R}$-linear maps ϵ, ∂_i $(i = 0, 1, 2)$. For $f^{(n)} \in \Sigma$, the homomorphism $\tilde{\partial}_0 : \mathbb{Z} \otimes_{\mathcal{R}} \mathcal{R}\underline{\mathbf{P}_1} \to \mathbb{Z} \otimes_{\mathcal{R}} \mathcal{R}\underline{\mathbf{P}_0}$ is given by

$$\tilde{\partial}_0(\underline{f}) = (n-1)\underline{1}.$$

For a term t, we define $\varphi(t)$ as the linear combinaton of symbols $\sum_{f\in\Sigma} n_f \underline{f}$ where n_f is the number of occurrences of f in t. Using this, for $l \to r \in R$, the homomorphism $\tilde{\partial}_1 : \mathbb{Z} \otimes_{\mathcal{R}} \mathcal{R}\underline{\mathbf{P}_2} \to \mathbb{Z} \otimes_{\mathcal{R}} \mathcal{R}\underline{\mathbf{P}_1}$ is given by

$$\tilde{\partial}_1(\underline{l \to r}) = \varphi(r) - \varphi(l).$$

For a critical pair $((\square, l \to r, s), (C, u \to v, t))$, let $(D_i, l_i \to r_i, s_i)$, $(C_j, u_j \to v_j, t_j)$ $(i = 1, \dots, k, j = 1, \dots, l)$ be rewriting steps such that $r \circ s = D_1[l_1 \circ s_1], D_1[r_1 \circ s_1] = D_2[l_2 \circ s_2], \dots, D_{k-1}[r_{k-1} \circ s_{k-1}] = D_k[l_k \circ s_k]$, $C[v \circ t] = C_1[u_1 \circ t_1], C_1[v_1 \circ t_1] = C_2[u_2 \circ t_2], \dots, C_{l-1}[v_{l-1} \circ t_{l-1}] = C_l[u_l \circ t_l]$, $D_k[r_k \circ s_k] = C_l[v_l \circ t_l]$. Then the map $\tilde{\partial}_2((\square, l \to r, s), (C, u \to v, t))$ is defined by

$$\underline{u \to v} - \underline{l \to v} - \sum_{i=1}^{k} \underline{u_i \to v_i} - \sum_{j=1}^{l} \underline{l_j \to r_j}.$$

Malbos-Mimram's lower bound for the number of rewrite rules is given by $s(H_2(\Sigma, R))$. (Recall that $s(G)$ denotes the minimum number of generators of an abelian group G.) More precisely, $\#\Sigma' \geq s(H_1(\Sigma, R))$ and $\#R' \geq s(H_2(\Sigma, R))$ hold for any TRS (Σ', R') that is Tietze equivalent to (Σ, R). These inequalities are shown in a similar way to the proof of Theorem 25.

5 Proof of Main Theorem

Let (Σ, R) be a complete TRS. We first simplify the tensor product $\mathbb{Z} \otimes_{\mathcal{R}} \mathbb{Z}\underline{\mathbf{P}_i}$ in (8).

▶ **Lemma 31.** *Let $d = \deg(R)$ and P be a family of sets $P_0, P_1, \dots$. Then, we have $\mathbb{Z} \otimes_{\mathcal{R}} \mathcal{R}\underline{P} \cong (\mathbb{Z}/d\mathbb{Z})\underline{\biguplus_i P_i}$. Especially, if $d = 0$, $\mathbb{Z} \otimes_{\mathcal{R}} \mathcal{R}\underline{P} \cong \mathbb{Z}\underline{\biguplus_i P_i}$.*

Proof. We define a group homomorphism $f : \mathbb{Z} \otimes_{\mathcal{R}} \mathcal{R}\underline{P} \to (\mathbb{Z}/d\mathbb{Z})\underline{\biguplus_i P_i}$ by $f((w_n)_{n\geq 0}) = \sum_{n\geq 0} f_n(w_n)$ where $f_n : \mathbb{Z} \otimes_{\mathcal{R}(n,n)} \mathcal{R}\underline{P}(n) \to (\mathbb{Z}/d\mathbb{Z})\underline{P_n}$ is defined by $f_n(k \otimes C\underline{a}t) = [k]\underline{a}$ for $a \in P_n$. It is enough to show each f_n is an isomorphism. If $\#_i l - \#_i r = m$ for $l \to r \in R$, we have a relation of $\mathcal{R}$

$$0 = 1 \otimes (\kappa_i(l)\underline{a}t - \kappa_i(r)\underline{a}t) = 1 \otimes \kappa_i(l)\underline{a}t - 1 \otimes \kappa_i(r)\underline{a}t = \#_i l \otimes \underline{a} - \#_i r \otimes \underline{a} = m \otimes \underline{a}.$$

Since d divides m, $f_n(m \otimes \underline{a}) = [m]\underline{a} = 0$. Therefore f_n is well-defined. To prove f_n is injective, it suffices to show $qd \otimes \underline{a} = 0$ for any $q \in \mathbb{Z}$. Since $d = \gcd\{\#_i l - \#_i r \mid l \to r \in R, i = 1, 2, \dots\}$, there exist integers $c_{i,l\to r}$ such that $d = \sum_{l\to r\in R,\ i=1,2,\dots} c_{i,l\to r}(\#_i l - \#_i r)$. Since $(\#_i l - \#_i r) \otimes \underline{a} = 1 \otimes (\kappa_i(l) - \kappa_i(r))\underline{a}\langle x_1, \dots, x_n\rangle = 0$ for each $i \in \{0, 1, \dots\}$, $l \to r \in R$, we have $qd \otimes \underline{a} = q \sum_{l\to r\in R,\ i=1,2,\dots} c_{i,l\to r}(\#_i l - \#_i r) \otimes \underline{a} = 0$. The surjectivity of f_n is trivial. ◀

As special cases of this lemma, we have $\mathbb{Z} \otimes_{\mathcal{R}} \mathcal{R}\underline{\mathbf{P}_0} \cong (\mathbb{Z}/d\mathbb{Z})\underline{\Sigma}$, $\mathbb{Z} \otimes_{\mathcal{R}} \mathcal{R}\underline{\mathbf{P}_1} \cong (\mathbb{Z}/d\mathbb{Z})\underline{R}$, and $\mathbb{Z} \otimes_{\mathcal{R}} \mathcal{R}\underline{\mathbf{P}_2} \cong (\mathbb{Z}/d\mathbb{Z})\underline{\mathrm{CP}(R)}$. Additionally, we can see each group homomorphism $\tilde{\partial}_i$ $(i = 0, 1, 2)$ is a $\mathbb{Z}/d\mathbb{Z}$-linear map.

To prove Theorem 5, we show the following lemma.

▶ **Lemma 32.** *Let $d = \deg(R)$. If $d = 0$ or d is prime, $\#R - e(R) = s(H_2(\Sigma, R)) + s(\operatorname{im}\tilde{\partial}_1)$.*

Proof. By definition, $D(R)$ defined in Section 2 is a matrix representation of $\tilde{\partial}_2$. Suppose d is prime. In this case, $s(H_2(\Sigma, R))$ is equal to the dimension of $H_2(\Sigma, R)$ as a $\mathbb{Z}/d\mathbb{Z}$-vector space. By the rank-nullity theorem, we have

$$\begin{aligned}
\dim(H_2(\Sigma, R)) &= \dim(\ker\tilde{\partial}_1) - \dim(\operatorname{im}\tilde{\partial}_2)\\
&= \dim(\mathbb{Z} \otimes_{\mathcal{R}} \mathcal{R}\underline{\mathbf{P}_1}) - \dim(\operatorname{im}\tilde{\partial}_1) - \dim(\operatorname{im}\tilde{\partial}_2)\\
&= \dim((\mathbb{Z}/d\mathbb{Z})\underline{R}) - \dim(\operatorname{im}\tilde{\partial}_1) - \operatorname{rank}(D(R))\\
&= \#R - \dim(\operatorname{im}\tilde{\partial}_1) - e(R).
\end{aligned}$$

Suppose $d = 0$. We show $H_2(\Sigma, R) \cong \mathbb{Z}^{\#R-r-k} \times \mathbb{Z}/e_1\mathbb{Z} \times \cdots \times \mathbb{Z}/e_r\mathbb{Z}$ where $r = \operatorname{rank}(D(R))$, $k = s(\operatorname{im}\tilde{\partial}_1)$, and $e_1, \ldots, e_r$ are the elementary divisors of $D(R)$. Let

$$\overline{\partial}_1 : \mathbb{Z} \otimes_{\mathcal{R}} \mathcal{R}\underline{\mathbf{P}_1}/\operatorname{im}\tilde{\partial}_2 \to \mathbb{Z} \otimes_{\mathcal{R}} \mathcal{R}\underline{\mathbf{P}_0}$$

be the group homomorphism defined by $[x] \mapsto \tilde{\partial}_1(x)$. $\overline{\partial}_1$ is well-defined since $\operatorname{im}\tilde{\partial}_2 \subset \ker\tilde{\partial}_1$, and $\ker\overline{\partial}_1$ is isomorphic to $\ker\tilde{\partial}_1/\operatorname{im}\tilde{\partial}_2 = H_2(\Sigma, R)$. By taking the basis $v_1, \ldots, v_{\#R}$ of $\mathbb{Z}\otimes_{\mathcal{R}}\mathcal{R}\underline{\mathbf{P}_1} \cong \mathbb{Z}\underline{R}$ such that $D(R)$ is the matrix representation of $\tilde{\partial}_2$ under the basis $v_1, \ldots, v_{\#R}$ and some basis of $\mathbb{Z} \otimes_{\mathcal{R}} \mathcal{R}\underline{\mathbf{P}_2}$, we can see $\mathbb{Z} \otimes_{\mathcal{R}} \mathcal{R}\underline{\mathbf{P}_1}/\operatorname{im}\tilde{\partial}_2 \cong \mathbb{Z}^{\#R-r} \times \mathbb{Z}/e_1\mathbb{Z} \times \cdots \times \mathbb{Z}/e_k\mathbb{Z}$. Suppose $\overline{\partial}_1(e_i[x]) = 0$ for some x and $i = 1, \ldots, r$. Since $\overline{\partial}_1$ is a homomorphism, $\overline{\partial}_1(e_i[x]) = e_i\overline{\partial}_1([x]) \in \mathbb{Z} \otimes_{\mathcal{R}} \mathcal{R}\underline{\mathbf{P}_0} \cong \mathbb{Z}\underline{\Sigma}$ holds. Since $\mathbb{Z}\underline{\Sigma}$ is free, we have $[x] = 0$. Therefore, $\ker\overline{\partial}_1$ is included in the subset of $\mathbb{Z} \otimes_{\mathcal{R}} \mathcal{R}\underline{\mathbf{P}_1}/\operatorname{im}\tilde{\partial}_2$ isomorphic to $\mathbb{Z}^{\#R-r} \times \{0\} \times \cdots \times \{0\}$. Thus, $\ker\overline{\partial}_1 \cong \mathbb{Z}^{\#R-r-k} \times \mathbb{Z}/e_1\mathbb{Z} \times \cdots \times \mathbb{Z}/e_r\mathbb{Z}$.

Since $\mathbb{Z}/e\mathbb{Z} \cong 0$ if e is invertible, $\mathbb{Z}^{\#R-r-k} \times \mathbb{Z}/e_1\mathbb{Z} \times \cdots \times \mathbb{Z}/e_k\mathbb{Z} \cong \mathbb{Z}^{\#R-r-k} \times \mathbb{Z}/e_{e(R)+1}\mathbb{Z} \times \cdots \mathbb{Z}/e_r\mathbb{Z} =: G$. G is generated by $(\underbrace{1, 0, \ldots, 0}_{\#R-r-k}, \underbrace{[0], \ldots, [0]}_{r-e(R)})$, $(0, \ldots, 0, 1, [0], \ldots, [0])$, $(0, \ldots, 0, [1], [0], \ldots, [0])$, $(0, \ldots, 0, [0], \ldots, [0], [1])$, so we have $s(G) \leq \#R - r - k + r - e(R) = \#R - k - e(R)$. Let p be a prime number which divides $e_{e(R)+1}$. We can see $G/pG \cong (\mathbb{Z}/p\mathbb{Z})^{\#R-k-e(R)}$. It is not hard to see $s(G) \geq s(G/pG)$, and since G/pG is a $\mathbb{Z}/p\mathbb{Z}$-vector space, $s(G/pG) = \dim(G/pG) = \#R - k - e(R)$. Thus, $s(H_2(\Sigma, R)) = s(G) = \#R - s(\operatorname{im}\tilde{\partial}_1) - e(R)$. ◀

By Lemma 32, Theorem 5 is implied by the following theorem:

▶ **Theorem 33.** *Let (Σ, R) be a complete TRS and $d = \deg(R)$. If $d = 0$ or d is prime,*

$$\#R \geq s(H_2(\Sigma, R)) + s(\operatorname{im}\tilde{\partial}_1). \tag{9}$$

Proof. By the first isomorphism theorem, we have an isomorphism between $\mathbb{Z}/d\mathbb{Z}$-modules

$$\operatorname{im}\tilde{\partial}_1 \simeq \mathbb{Z} \otimes_{\mathcal{R}} \mathcal{R}\underline{\mathbb{P}_2}/\ker\tilde{\partial}_1$$

and by the third isomorphism theorem, the right hand side is isomorphic to

$$\begin{aligned}
\mathbb{Z} \otimes_{\mathcal{R}} \mathcal{R}\underline{\mathbb{P}_2}/\ker\tilde{\partial}_1 &\simeq \big(\mathbb{Z} \otimes_{\mathcal{R}} \mathcal{R}\underline{\mathbb{P}_2}/\operatorname{im}\tilde{\partial}_2\big)/\big(\ker\tilde{\partial}_1/\operatorname{im}\tilde{\partial}_2\big)\\
&\simeq \big(\mathbb{Z} \otimes_{\mathcal{R}} \mathcal{R}\underline{\mathbb{P}_2}/\operatorname{im}\tilde{\partial}_2\big)/H_2(\Sigma, R).
\end{aligned}$$

Thus, we obtain the following exact sequence by Proposition 23:

$$0 \to H_2(\Sigma, R) \to \mathbb{Z} \otimes_{\mathcal{R}} \mathcal{R}\underline{\mathbb{P}_2}/\mathrm{im}\,\tilde{\partial}_2 \to \mathrm{im}\,\tilde{\partial}_1 \to 0.$$

By Theorem 17, since $\mathrm{im}\,\tilde{\partial}_1 \subset \mathbb{Z} \otimes_{\mathcal{R}} \mathcal{R}\underline{\mathbb{P}_1} \cong (\mathbb{Z}/d\mathbb{Z})\underline{R}$ and $(\mathbb{Z}/d\mathbb{Z})\underline{R}$ is a free $\mathbb{Z}/d\mathbb{Z}$-module, $\mathrm{im}\,\tilde{\partial}_1$ is also free and by Proposition 24, we have $\mathbb{Z} \otimes_{\mathcal{R}} \mathcal{R}\underline{\mathbb{P}_2}/\mathrm{im}\,\tilde{\partial}_2 \cong H_2(\Sigma, R) \times \mathrm{im}\,\tilde{\partial}_1$. Therefore, $s(\mathbb{Z} \otimes_{\mathcal{R}} \mathcal{R}\underline{\mathbb{P}_2}/\mathrm{im}\,\tilde{\partial}_2) = s(H_2(\Sigma, R)) + s(\mathrm{im}\,\tilde{\partial}_1)$. Since $\mathbb{Z} \otimes_{\mathcal{R}} \mathcal{R}\underline{\mathbb{P}_2}/\mathrm{im}\,\tilde{\partial}_2$ is generated by $[l_1 \to r_1], \ldots, [l_k \to r_k]$ if $R = \{l_1 \to r_1, \ldots, l_k \to r_k\}$, we obtain

$$k = \#R \geq s(\mathbb{Z} \otimes_{\mathcal{R}} \mathcal{R}\underline{\mathbb{P}_2}/\mathrm{im}\,\tilde{\partial}_2) = s(H_2(\Sigma, R)) + s(\mathrm{im}\,\tilde{\partial}_1).$$

Thus, we get (9). ◀

Proof of Theorem 5. As we stated, $H_2(\Sigma, R)$ depends only on the Tietze equivalence class of (Σ, R). Let us show $s(\mathrm{im}\,\tilde{\partial}_1)$ also depends only on the Tietze equivalence class of (Σ, R). For a left $\mathfrak{R}$-module M, $\mathrm{rank}(M)$ denotes the cardinality of a minimal linearly independent generating set of M, that is, a minimal generating set S of G such that any element $s_1, \ldots, s_k \in \Gamma$, and $r_1 s_1 + \cdots + r_k s_k = 0 \implies r_1 = \cdots = r_k = 0$ for any $r_1, \ldots, r_k \in \mathfrak{R}$, $s_1, \ldots, s_k \in S$. It can be shown that $\mathrm{rank}(M) = s(M)$ if M is free. Especially, $s(\mathrm{im}\,\tilde{\partial}_1) = \mathrm{rank}(\mathrm{im}\,\tilde{\partial}_1)$ since $\mathrm{im}\,\tilde{\partial}_1 \subset \mathbb{Z}\underline{R}$ if $\deg(R) = 0$. Also, $\mathrm{rank}(\mathrm{im}\,\tilde{\partial}_1) = \mathrm{rank}(\ker\tilde{\partial}_0) - \mathrm{rank}(\ker\tilde{\partial}_0/\,\mathrm{im}\,\tilde{\partial}_1)$ is obtained by a general theorem [7, Ch 10, Lemma 10.1]. By definition, $\tilde{\partial}_0$ does not depend on R. Since $\ker\tilde{\partial}_0/\,\mathrm{im}\,\tilde{\partial}_1 = H_1(\Sigma, R)$ depends only on the Tietze equivalence class of (Σ, R), so does $\mathrm{rank}(\mathrm{im}\,\tilde{\partial}_1)$.

In conclusion, for any TRS R' equivalent to R, we obtain $\#R' \geq s(H_2(\Sigma, R)) + s(\mathrm{im}\,\tilde{\partial}_1) = \#R - e(R)$. ◀

We consider the case where every symbol in Σ is of arity 1. Notice that any TRS (Σ, R) can be seen as an SRS and $\deg(R) = 0$ in this case. We have $\mathrm{rank}(\ker\tilde{\partial}_0) = \#\Sigma$ since $\tilde{\partial}_0(\underline{f}) = 0$ for any $f \in \Sigma$. Therefore, (9) can be rewritten to

$$\#R - \#\Sigma \geq s(H_2(\Sigma, R)) - \mathrm{rank}(H_1(\Sigma, R)).$$

So, for SRSs, we have a lower bound of the number of the rewrite rules minus the number of the symbols. For groups, in fact, this inequality is proved in terms of group homology [1].

References

1. D. Epstein. Finite presentations of groups and 3-manifolds. *The Quarterly Journal of Mathematics*, 12(1):205–212, 1961.
2. K. Kunen. Single axioms for groups. *Journal of Automated Reasoning*, 9(3):291–308, December 1992. doi:10.1007/BF00245293.
3. P. Malbos and S. Mimram. Homological computations for term rewriting systems. In *1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016)*, volume 52 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 27:1–27:17, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
4. B. H. Neumann. Another single law for groups. *Bulletin of the Australian Mathematical Society*, 23(1):81–102, 1981. doi:10.1017/S0004972700006912.
5. B. H. Neumann. Yet another single law for groups. *Illinois J. Math.*, 30(2):295–300, June 1986.
6. J. J. Rotman. *An Introduction to Homological Algebra*. Springer-Verlag New York, 2009.
7. J. J. Rotman. *Advanced Modern Algebra*, volume 114. American Mathematical Soc., 2010.
8. C. C. Squier. Word problems and a homological finiteness condition for monoids. *Journal of Pure and Applied Algebra*, 49(1-2):201–217, 1987.

9 A. Tarski. Equational Logic and Equational Theories of Algebras. In *Contributions to Mathematical Logic*, volume 50 of *Studies in Logic and the Foundations of Mathematics*, pages 275–288. Elsevier, 1968.

A The matrix $D(R)$ for The Theory of Groups

For the TRS R defined in Example 7, $D(R)$ is given by the transpose of

$$\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0
\end{pmatrix}$$

where the i-th column corresponds to the rule G_i, and the j-th row corresponds to the critical pair C_j shown in the next pages.

$C_1 : m(m(x_1,x_2),x_3) \to m(x_1,m(x_2,x_3)), \quad m(m(x_4,x_5),x_6) \to m(x_4,m(x_5,x_6)), \quad m(\Box,x_3),$
$\{x_6 \mapsto x_2, x_1 \mapsto m(x_4,x_5)\}$

$C_2 : i(m(x_1,x_2)) \to m(i(x_2),i(x_1)), \quad m(m(x_3,x_4),x_5) \to m(x_3,m(x_4,x_5)), \quad i(\Box),$
$\{x_5 \mapsto x_2, x_1 \mapsto m(x_3,x_4)\}$

$C_3 : m(m(x_1,x_2),x_3) \to m(x_1,m(x_2,x_3)), \quad m(x_4,m(i(x_4),x_5)) \to x_5, \quad m(\Box,x_3),$
$\{x_2 \mapsto m(i(x_1),x_5), x_4 \mapsto x_1\}$

$C_4 : m(x_1,m(i(x_1),x_2)) \to x_2, \quad m(m(x_3,x_4),x_5) \to m(x_3,m(x_4,x_5)), \quad \Box,$
$\{x_5 \mapsto m(i(m(x_3,x_4)),x_2), x_1 \mapsto m(x_3,x_4)\}$

$C_5 : m(m(x_1,x_2),x_3) \to m(x_1,m(x_2,x_3)), \quad m(i(x_4),m(x_4,x_5)) \to x_5, \quad m(\Box,x_3),$
$\{x_2 \mapsto m(x_4,x_5), x_1 \mapsto i(x_4)\}$

$C_6 : m(i(x_1),m(x_1,x_2)) \to x_2, \quad m(m(x_3,x_4),x_5) \to m(x_3,m(x_4,x_5)), \quad m(i(x_1),\Box),$
$\{x_5 \mapsto x_2, x_1 \mapsto m(x_3,x_4)\}$

$C_7 : m(m(x_1,x_2),x_3) \to m(x_1,m(x_2,x_3)), \quad m(i(x_4),x_4) \to e, \quad m(\Box,x_3),$
$\{x_4 \mapsto x_2, x_1 \mapsto i(x_2)\}$

$C_8 : m(m(x_1,x_2),x_3) \to m(x_1,m(x_2,x_3)), \quad m(x_4,i(x_4)) \to e, \quad m(\Box,x_3),$
$\{x_2 \mapsto i(x_1), x_4 \mapsto x_1\}$

$C_9 : m(x_1,i(x_1)) \to e, \quad m(m(x_2,x_3),x_4) \to m(x_2,m(x_3,x_4)), \quad \Box,$
$\{x_4 \mapsto i(m(x_2,x_3)), x_1 \mapsto m(x_2,x_3)\}$

$C_{10} : m(m(x_1,x_2),x_3) \to m(x_1,m(x_2,x_3)), \quad m(x_4,e) \to x_4, \quad m(\Box,x_3), \quad \{x_2 \mapsto e, x_4 \mapsto x_1\}$

$C_{11} : m(x_1,e) \to x_1, \quad m(m(x_2,x_3),x_4) \to m(x_2,m(x_3,x_4)), \quad \Box, \quad \{x_4 \mapsto e, x_1 \mapsto m(x_2,x_3)\}$

$C_{12} : m(m(x_1,x_2),x_3) \to m(x_1,m(x_2,x_3)), \quad m(e,x_4) \to x_4, \quad m(\Box,x_3), \quad \{x_4 \mapsto x_2, x_1 \mapsto e\}$

$C_{13} : i(m(x_1,x_2)) \to m(i(x_2),i(x_1)), \quad m(e,x_3) \to x_3, \quad i(\Box), \quad \{x_3 \mapsto x_2, x_1 \mapsto e\}$

$C_{14} : m(x_1,m(i(x_1),x_2)) \to x_2, \quad m(e,x_3) \to x_3, \quad \Box, \quad \{x_3 \mapsto m(i(e),x_2), x_1 \mapsto e\}$

$C_{15} : m(i(x_1),m(x_1,x_2)) \to x_2, \quad m(e,x_3) \to x_3, \quad m(i(x_1),\Box), \quad \{x_3 \mapsto x_2, x_1 \mapsto e\}$

$C_{16} : m(x_1,i(x_1)) \to e, \quad m(e,x_2) \to x_2, \quad \Box, \quad \{x_2 \mapsto i(e), x_1 \mapsto e\}$

$C_{17} : m(x_1,e) \to x_1, \quad m(e,x_2) \to x_2, \quad \Box, \quad \{x_2 \mapsto e, x_1 \mapsto e\}$

$C_{18} : i(m(x_1,x_2)) \to m(i(x_2),i(x_1)), \quad m(x_3,e) \to x_3, \quad i(\Box), \quad \{x_2 \mapsto e, x_3 \mapsto x_1\}$

$C_{19} : m(x_1,m(i(x_1),x_2)) \to x_2, \quad m(x_3,e) \to x_3, \quad m(x_1,\Box), \quad \{x_2 \mapsto e, x_3 \mapsto i(x_1)\}$

$C_{20} : m(i(x_1),m(x_1,x_2)) \to x_2, \quad m(x_3,e) \to x_3, \quad m(i(x_1),\Box), \quad \{x_2 \mapsto e, x_3 \mapsto x_1\}$

$C_{21} : m(i(x_1),x_1) \to e, \quad m(x_2,e) \to x_2, \quad \Box, \quad \{x_1 \mapsto e, x_2 \mapsto i(e)\}$

$C_{22} : m(x_1,i(x_1)) \to e, \quad i(m(x_2,x_3)) \to m(i(x_3),i(x_2)), \quad m(x_1,\Box), \quad \{x_1 \mapsto m(x_2,x_3)\}$

$C_{23} : i(m(x_1,x_2)) \to m(i(x_2),i(x_1)), \quad m(x_3,i(x_3)) \to e, \quad i(\Box), \quad \{x_2 \mapsto i(x_1), x_3 \mapsto x_1\}$

$C_{24} : m(x_1,m(i(x_1),x_2)) \to x_2, \quad m(x_3,i(x_3)) \to e, \quad m(x_1,\Box), \quad \{x_2 \mapsto i(i(x_1)), x_3 \mapsto i(x_1)\}$

Figure 2 The critical pairs of the complete TRS R
($C_j :\ l \to r,\ l' \to r',\ C,\ \sigma$ means C_j is the critical pair $(r\sigma, C[r'\sigma])$.) – Part 1.

$C_{25}: m(x_1, i(x_1)) \to e, \quad i(i(x_2)) \to x_2, \quad m(x_1, \square), \quad \{x_1 \mapsto i(x_2)\}$

$C_{26}: m(x_1, i(x_1)) \to e, \quad i(e) \to e, \quad m(x_1, \square), \quad \{x_1 \mapsto e\}$

$C_{27}: m(i(x_1), m(x_1, x_2)) \to x_2, \quad m(x_3, i(x_3)) \to e, \quad m(i(x_1), \square), \quad \{x_2 \mapsto i(x_1), x_3 \mapsto x_1\}$

$C_{28}: m(i(x_1), x_1) \to e, \quad i(m(x_2, x_3)) \to m(i(x_3), i(x_2)), \quad m(\square, x_1), \quad \{x_1 \mapsto m(x_2, x_3)\}$

$C_{29}: i(m(x_1, x_2)) \to m(i(x_2), i(x_1)), \quad m(i(x_3), x_3) \to e, \quad i(\square), \quad \{x_3 \mapsto x_2, x_1 \mapsto i(x_2)\}$

$C_{30}: m(x_1, m(i(x_1), x_2)) \to x_2, \quad m(i(x_3), x_3) \to e, \quad m(x_1, \square), \quad \{x_1 \mapsto x_2, x_3 \mapsto x_2\}$

$C_{31}: m(i(x_1), x_1) \to e, \quad i(i(x_2)) \to x_2, \quad m(\square, x_1), \quad \{x_1 \mapsto i(x_2)\}$

$C_{32}: m(i(x_1), x_1) \to e, \quad i(e) \to e, \quad m(\square, x_1), \quad \{x_1 \mapsto e\}$

$C_{33}: m(i(x_1), m(x_1, x_2)) \to x_2, \quad m(i(x_3), x_3) \to e, \quad m(i(x_1), \square), \quad \{x_3 \mapsto x_2, x_1 \mapsto i(x_2)\}$

$C_{34}: m(i(x_1), m(x_1, x_2)) \to x_2, \quad m(i(x_3), m(x_3, x_4)) \to x_4, \quad m(i(x_1), \square), \quad \{x_2 \mapsto m(x_3, x_4), x_1 \mapsto i(x_3)\}$

$C_{35}: m(i(x_1), m(x_1, x_2)) \to x_2, \quad i(m(x_3, x_4)) \to m(i(x_4), i(x_3)), \quad m(\square, m(x_1, x_2)), \quad \{x_1 \mapsto m(x_3, x_4)\}$

$C_{36}: i(m(x_1, x_2)) \to m(i(x_2), i(x_1)), \quad m(i(x_3), m(x_3, x_4)) \to x_4, \quad i(\square), \quad \{x_2 \mapsto m(x_3, x_4), x_1 \mapsto i(x_3)\}$

$C_{37}: m(i(x_1), m(x_1, x_2)) \to x_2, \quad m(x_3, m(i(x_3), x_4)) \to x_4, \quad m(i(x_1), \square), \quad \{x_2 \mapsto m(i(x_1), x_4), x_3 \mapsto x_1\}$

$C_{38}: m(x_1, m(i(x_1), x_2)) \to x_2, \quad m(i(x_3), m(x_3, x_4)) \to x_4, \quad m(x_1, \square), \quad \{x_2 \mapsto m(x_1, x_4), x_3 \mapsto x_1\}$

$C_{39}: m(i(x_1), m(x_1, x_2)) \to x_2, \quad i(i(x_3)) \to x_3, \quad m(\square, m(x_1, x_2)), \quad \{x_1 \mapsto i(x_3)\}$

$C_{40}: m(i(x_1), m(x_1, x_2)) \to x_2, \quad i(e) \to e, \quad m(\square, m(x_1, x_2)), \quad \{x_1 \mapsto e\}$

$C_{41}: m(x_1, m(i(x_1), x_2)) \to x_2, \quad i(e) \to e, \quad m(x_1, m(\square, x_2)), \quad \{x_1 \mapsto e\}$

$C_{42}: i(i(x_1)) \to x_1, \quad i(e) \to e, \quad i(\square), \quad \{x_1 \mapsto e\}$

$C_{43}: i(i(x_1)) \to x_1, \quad i(i(x_2)) \to x_2, \quad i(\square), \quad \{x_1 \mapsto i(x_2)\}$

$C_{44}: i(i(x_1)) \to x_1, \quad i(m(x_2, x_3)) \to m(i(x_3), i(x_2)), \quad i(\square), \quad \{x_1 \mapsto m(x_2, x_3)\}$

$C_{45}: m(x_1, m(i(x_1), x_2)) \to x_2, \quad i(i(x_3)) \to x_3, \quad m(x_1, m(\square, x_2)), \quad \{x_1 \mapsto i(x_3)\}$

$C_{46}: m(x_1, m(i(x_1), x_2)) \to x_2, \quad m(x_3, m(i(x_3), x_4)) \to x_4, \quad m(x_1, \square),$
$\quad \{x_2 \mapsto m(i(i(x_1)), x_4), x_3 \mapsto i(x_1)\}$

$C_{47}: m(x_1, m(i(x_1), x_2)) \to x_2, \quad i(m(x_3, x_4)) \to m(i(x_4), i(x_3)), \quad m(x_1, m(\square, x_2)), \quad \{x_1 \mapsto m(x_3, x_4)\}$

$C_{48}: i(m(x_1, x_2)) \to m(i(x_2), i(x_1)), \quad m(x_3, m(i(x_3), x_4)) \to x_4, \quad i(\square), \quad \{x_2 \mapsto m(i(x_1), x_4), x_3 \mapsto x_1\}$

Figure 3 The critical pairs of the complete TRS R
($C_j: \ l \to r, \ l' \to r', \ C, \ \sigma$ means C_j is the critical pair $(r\sigma, C[r'\sigma])$.) – Part 2.

Gluing for Type Theory

Ambrus Kaposi
Eötvös Loránd University, Budapest, Hungary
akaposi@inf.elte.hu

Simon Huber
University of Gothenburg, Sweden
simonhu@chalmers.se

Christian Sattler
University of Gothenburg, Sweden
sattler@chalmers.se

Abstract

The relationship between categorical gluing and proofs using the logical relation technique is folklore. In this paper we work out this relationship for Martin-Löf type theory and show that parametricity and canonicity arise as special cases of gluing. The input of gluing is two models of type theory and a pseudomorphism between them and the output is a displayed model over the first model. A pseudomorphism preserves the categorical structure strictly, the empty context and context extension up to isomorphism, and there are no conditions on preservation of type formers. We look at three examples of pseudomorphisms: the identity on the syntax, the interpretation into the set model and the global section functor. Gluing along these result in syntactic parametricity, semantic parametricity and canonicity, respectively.

2012 ACM Subject Classification Theory of computation → Type theory

Keywords and phrases Martin-Löf type theory, logical relations, parametricity, canonicity, quotient inductive types

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.25

Funding *Ambrus Kaposi*: The author was supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013, Thematic Fundamental Research Collaborations Grounding Innovation in Informatics and Infocommunications) and COST Action EUTypes CA15123.
Simon Huber: I acknowledge the support of the Centre for Advanced Study (CAS) in Oslo, Norway, which funded and hosted the research project Homotopy Type Theory and Univalent Foundations during the academic year 2018/19.

Acknowledgements The authors thank Thorsten Altenkirch, Simon Boulier, Thierry Coquand, András Kovács and Nicolas Tabareau for discussions related to the topics of this paper.

1 Introduction

Categorical gluing [11, Section 4.10] is a method to form a new category from two categories and a functor between them. This goes back to the Artin gluing of Grothendieck toposes [5, Exposé IV, Section 9.5]. Given a functor F from category $\mathcal{S}$ to $\mathcal{M}$, an object in the glued category is a triple $\Gamma : |\mathcal{S}|$, $\Delta : |\mathcal{M}|$ and a morphism $\mathcal{M}(\Delta, F\,\Gamma)$. Models of logics and type theories can be given as categories with extra structure and gluing can be extended to these models. Gluing was used to prove properties of closed proofs in intuitionistic higher-order logic [21] and normalisation for simple type theory [14, 26] and System F [2]. In programming language semantics, similar results are proved more syntactically using the technique of logical relations [22, 24], see [15] for an introduction and [13, 1] for more recent proofs using this technique. It is folklore that logical relations correspond to gluing. Logical relations scale to real-world systems [27, 19] while gluing is a more abstract construction which can be applied to systems with well-understood categorical semantics.

© Ambrus Kaposi, Simon Huber, and Christian Sattler;
licensed under Creative Commons License CC-BY

In this paper we develop the correspondence between proof-relevant logical predicates and gluing for Martin-Löf type theory. Logical relations were defined for type theory to prove free theorems in syntactic [7] and semantic (Reynolds-style) [6] ways. Proof-relevant logical predicates were employed to prove normalisation and canonicity for type theory [4, 10, 20]. We unify these approaches by defining gluing in an abstract way, for any pseudomorphism between two models of type theory. An important characteristic of our approach is using an algebraic syntax of type theory. By this we mean the well-typed syntax of type theory given as a quotient inductive-inductive type (QIIT, [18]). A model of this syntax is just an algebra of the QIIT which turns out to be the same as a category with families (CwF, [12]) with extra structure. A pseudomorphism of models is a map from sorts in one model to sorts in the other model which preserves the categorical structure strictly and the empty context and context extension up to isomorphism. We show that gluing can be performed along any pseudomoprhism and gluing preserves Π, Σ, Bool and an infinite hierarchy of universes.

Our motivational guideline for this paper is the following.

1. Gluing over identity is syntactic parametricity.
2. Gluing over the interpretation into the set model is semantic parametricity.
3. Gluing over the global section functor is canonicity.
4. Gluing over Yoneda is normalisation.
5. Gluing over Yoneda composed with the set interpretation is definability/completeness.

In this paper we only generalise steps 1–3. The Yoneda embedding (from the syntax to the presheaf model over a wide subcategory of contexts and substitutions) is also a pseudomorphism, so our paper applies to steps 4–5 as well. However, Yoneda has extra structure that we do not employ in this paper. This extra structure is needed to obtain full normalisation or completeness.

Structure of the paper

After summarizing related work and the metatheory, we define our object type theory in Section 2 and as an example we define its set model (Section 3). Then we define the notion of pseudomorphism (Section 4) and gluing for any pseudomorphism (Section 5). Afterwards, in Section 6 we define a non-trivial pseudomorphism: the global section functor which goes from the syntax to the set model and maps types to terms of the type in the empty context. We put together the pieces in Section 7 by obtaining parametricity and canonicity for our object theory using gluing. We conclude and summarize further work in Section 8.

Contribution

The contribution of this paper is showing that gluing can be defined for any pseudomorphism for Martin-Löf type theory. To our knowledge, this is the first general construction from which both parametricity and canonicity arise.

Related work

Sterling and Spitters [26] developed gluing for simple type theory and show how it relates to syntactic proofs of normalisation by logical relations and semantic proofs based on normalisation by evaluation. Altenkirch, Hofmann and Streicher developed gluing for System F and prove normalisation in their unpublished note [2]. Rabe and Sojakova [23] defined a syntactic framework for logical relations which applies to theories formulated in the Edinburgh

Logical Framework (LF). Shulman [25] developed gluing for type theory in the context of type-theoretic fibration categories and proves homotopy canonicity for a 1-truncated version of homotopy type theory. Compared to Shulman, we work with a notion of model closer to the syntax of type theory: categories with families. In previous work [4] we proved normalisation for type theory with Π, a base type and a base family. The logical predicate used in that proof is an instance of the abstract gluing technique presented in this paper. Coquand [10] proves canonicity and normalisation for a richer type theory with Bool and a hierarchy of universes. His canonicity proof is an unfolding of the canonicity proof given in this paper.

Metatheory and notation

Our metatheory is Martin-Löf's extensional type theory. We have a cumulative hierarchy of universes Set_0, Set_1, ... with Set_ω on top. Sometimes we omit the universe indices. Function space is denoted by $\rightarrow$ with constructor λ and application written as juxtaposition. We use implicit arguments extensively, e.g. we would write the type of function composition as $(B \rightarrow C) \rightarrow (A \rightarrow B) \rightarrow (A \rightarrow C)$ instead of $(A : \mathsf{Set}) \rightarrow (B : \mathsf{Set}) \rightarrow (C : \mathsf{Set}) \rightarrow (B \rightarrow C) \rightarrow (A \rightarrow B) \rightarrow (A \rightarrow C)$. When a metavariable is not quantified explicitly (such as A, B, C), we assume implicit quantification and implicit application as well. Sometimes we omit explicit arguments for readability, in this case we write underscore _ instead of the argument. Pairs are denoted by $\times$ with constructor $-,-$ and destructors $._1$ and $._2$. Both $\rightarrow$ and $\times$ come with η laws. The one-element type is denoted $\mathbb{1}$ with constructor $*$, the two-element type is denoted $\mathbb{2}$, its constructors being $*$ and $**$ and its eliminator case. Equality is denoted $=$ and we use equational reasoning to write equality proofs. We use equality reflection in the metatheory (not in our object theory described in Section 2). Following Hofmann [16], our arguments can be translated to an intensional type theory with function extensionality and uniqueness of identity proofs.

2 Type theory

By *type theory* we mean the (generalised) algebraic structure in Figure 1 with four sorts, 26 operators, and 34 equations. The four sorts are those of contexts, types, substitutions, and terms. Contexts and types are indexed by a universe level which is a metatheoretic natural number. Furthermore, types are indexed by contexts and terms by a context and a type in that context so that we can only mention well-typed terms. Substitutions are indexed by their domain and codomain, both contexts.

We explain the operators and laws for the substitution calculus (first column, operators id to $,\circ$) as follows: Con and Sub form a category (id to idr); there is a contravariant, functorial action of substitutions on types and terms ($-[-]$ to $[\circ]$), thus types (of fixed level) form a presheaf on the category and terms form a presheaf on its category of elements; there is an empty context $\cdot$ with a unique ($\cdot\eta$) empty substitution ϵ into it, thus $\cdot$ is the terminal object of the category; extended contexts can be formed using $- \triangleright -$ and there is a natural isomorphism between substitutions into $\Delta \triangleright A$ and a pair of a substitution σ into Δ and a term of type $A[\sigma]$ ($-,-$ to $,\circ$). The substitution calculus is the same as the structure of a category with families (CwF, [12]) with contexts and types indexed over natural numbers representing universe levels. In the CwF language, context extension is called comprehension. We denote n-fold iteration of the weakening substitution p by p^n (where $\mathsf{p}^0 = \mathsf{id}$), and we denote De Bruijn indices by natural numbers, i.e. $0 := \mathsf{q}$, $1 := \mathsf{q}[\mathsf{p}]$, ..., $n := \mathsf{q}[\mathsf{p}^n]$. We define

$\mathsf{Con} : \mathbb{N} \to \mathsf{Set}$
$\mathsf{Ty} : \mathbb{N} \to \mathsf{Con}\, i \to \mathsf{Set}$
$\mathsf{Sub} : \mathsf{Con}\, i \to \mathsf{Con}\, j \to \mathsf{Set}$
$\mathsf{Tm} : (\Gamma : \mathsf{Con}\, i) \to \mathsf{Ty}\, j\, \Gamma \to \mathsf{Set}$
$\mathsf{id} : \mathsf{Sub}\, \Gamma\, \Gamma$
$- \circ - : \mathsf{Sub}\, \Theta\, \Delta \to \mathsf{Sub}\, \Gamma\, \Theta \to \mathsf{Sub}\, \Gamma\, \Delta$
$\mathsf{ass} : (\sigma \circ \delta) \circ \nu = \sigma \circ (\delta \circ \nu)$
$\mathsf{idl} : \mathsf{id} \circ \sigma = \sigma$
$\mathsf{idr} : \sigma \circ \mathsf{id} = \sigma$
$-[-] : \mathsf{Ty}\, i\, \Delta \to \mathsf{Sub}\, \Gamma\, \Delta \to \mathsf{Ty}\, i\, \Gamma$
$-[-] : \mathsf{Tm}\, \Delta\, A \to (\sigma : \mathsf{Sub}\, \Gamma\, \Delta) \to \mathsf{Tm}\, \Gamma\, (A[\sigma])$
$[\mathsf{id}] : A[\mathsf{id}] = A$
$[\circ] : A[\sigma \circ \delta] = A[\sigma][\delta]$
$[\mathsf{id}] : t[\mathsf{id}] = t$
$[\circ] : t[\sigma \circ \delta] = t[\sigma][\delta]$
$\cdot : \mathsf{Con}\, 0$
$\epsilon : \mathsf{Sub}\, \Gamma\, \cdot$
$\cdot\eta : (\sigma : \mathsf{Sub}\, \Gamma\, \cdot) = \epsilon$
$- \triangleright - : (\Gamma : \mathsf{Con}\, i) \to \mathsf{Ty}\, j\, \Gamma \to \mathsf{Con}\, (i \sqcup j)$
$-,- : (\sigma : \mathsf{Sub}\, \Gamma\, \Delta) \to \mathsf{Tm}\, \Gamma\, (A[\sigma]) \to \mathsf{Sub}\, \Gamma\, (\Delta \triangleright A)$
$\mathsf{p} : \mathsf{Sub}\, (\Gamma \triangleright A)\, \Gamma$
$\mathsf{q} : \mathsf{Tm}\, (\Gamma \triangleright A)\, (A[\mathsf{p}])$
$\triangleright\beta_1 : \mathsf{p} \circ (\sigma, t) = \sigma$
$\triangleright\beta_2 : \mathsf{q}[\sigma, t] = t$
$\triangleright\eta : (\mathsf{p}, \mathsf{q}) = \mathsf{id}$
$,\circ : (\sigma, t) \circ \nu = (\sigma \circ \nu, t[\nu])$
$\Pi : (A : \mathsf{Ty}\, i\, \Gamma) \to \mathsf{Ty}\, j\, (\Gamma \triangleright A) \to \mathsf{Ty}\, (i \sqcup j)\, \Gamma$
$\mathsf{lam} : \mathsf{Tm}\, (\Gamma \triangleright A)\, B \to \mathsf{Tm}\, \Gamma\, (\Pi\, A\, B)$
$\mathsf{app} : \mathsf{Tm}\, \Gamma\, (\Pi\, A\, B) \to \mathsf{Tm}\, (\Gamma \triangleright A)\, B$
$\Pi\beta : \mathsf{app}\, (\mathsf{lam}\, t) = t$
$\Pi\eta : \mathsf{lam}\, (\mathsf{app}\, t) = t$
$\Pi[] : (\Pi\, A\, B)[\sigma] = \Pi\, (A[\sigma])\, (B[\sigma^{\uparrow}])$
$\mathsf{lam}[] : (\mathsf{lam}\, t)[\sigma] = \mathsf{lam}\, (t[\sigma^{\uparrow}])$

$\Sigma : (A : \mathsf{Ty}\, i\, \Gamma) \to \mathsf{Ty}\, j\, (\Gamma \triangleright A) \to \mathsf{Ty}\, (i \sqcup j)\, \Gamma$
$-,- : (u : \mathsf{Tm}\, \Gamma\, A) \to \mathsf{Tm}\, \Gamma\, (B[\mathsf{id}, u]) \to \mathsf{Tm}\, \Gamma\, (\Sigma\, A\, B)$
$\mathsf{projl} : \mathsf{Tm}\, \Gamma\, (\Sigma\, A\, B) \to \mathsf{Tm}\, \Gamma\, A$
$\mathsf{projr} : (t : \mathsf{Tm}\, \Gamma\, (\Sigma\, A\, B)) \to \mathsf{Tm}\, \Gamma\, (B[\mathsf{id}, \mathsf{projl}\, t])$
$\Sigma\beta_1 : \mathsf{projl}\, (u, v) = u$
$\Sigma\beta_2 : \mathsf{projr}\, (u, v) = v$
$\Sigma\eta : (\mathsf{projl}\, t, \mathsf{projr}\, t) = t$
$\Sigma[] : (\Sigma\, A\, B)[\sigma] = \Sigma\, (A[\sigma])\, (B[\sigma^{\uparrow}])$
$,[] : (u, v)[\sigma] = (u[\sigma], v[\sigma])$
$\top : \mathsf{Ty}\, 0\, \Gamma$
$\mathsf{tt} : \mathsf{Tm}\, \Gamma\, \top$
$\top\eta : (t : \mathsf{Tm}\, \Gamma\, \top) = \mathsf{tt}$
$\top[] : \top[\sigma] = \top$
$\mathsf{tt}[] : \mathsf{tt}[\sigma] = \mathsf{tt}$
$\mathsf{U} : (i : \mathbb{N}) \to \mathsf{Ty}\, (i+1)\, \Gamma$
$\mathsf{El} : \mathsf{Tm}\, \Gamma\, (\mathsf{U}\, i) \to \mathsf{Ty}\, i\, \Gamma$
$\mathsf{c} : \mathsf{Ty}\, i\, \Gamma \to \mathsf{Tm}\, \Gamma\, (\mathsf{U}\, i)$
$\mathsf{U}\beta : \mathsf{El}\, (\mathsf{c}\, A) = A$
$\mathsf{U}\eta : \mathsf{c}\, (\mathsf{El}\, a) = a$
$\mathsf{U}[] : (\mathsf{U}\, i)[\sigma] = (\mathsf{U}\, i)$
$\mathsf{El}[] : (\mathsf{El}\, a)[\sigma] = \mathsf{El}\, (a[\sigma])$
$\mathsf{Bool} : \mathsf{Ty}\, 0\, \Gamma$
$\mathsf{true} : \mathsf{Tm}\, \Gamma\, \mathsf{Bool}$
$\mathsf{false} : \mathsf{Tm}\, \Gamma\, \mathsf{Bool}$
$\mathsf{if} : (C : \mathsf{Ty}\, i\, (\Gamma \triangleright \mathsf{Bool})) \to \mathsf{Tm}\, \Gamma\, (C[\mathsf{id}, \mathsf{true}]) \to \mathsf{Tm}\, \Gamma\, (C[\mathsf{id}, \mathsf{false}]) \to (t : \mathsf{Tm}\, \Gamma\, \mathsf{Bool}) \to \mathsf{Tm}\, \Gamma\, (C[\mathsf{id}, t])$
$\mathsf{Bool}\beta_1 : \mathsf{if}\, C\, u\, v\, \mathsf{true} = u$
$\mathsf{Bool}\beta_2 : \mathsf{if}\, C\, u\, v\, \mathsf{false} = v$
$\mathsf{Bool}[] : \mathsf{Bool}[\sigma] = \mathsf{Bool}$
$\mathsf{true}[] : \mathsf{true}[\sigma] = \mathsf{true}$
$\mathsf{false}[] : \mathsf{false}[\sigma] = \mathsf{false}$
$\mathsf{if}[] : (\mathsf{if}\, C\, u\, v\, t)[\sigma] = \mathsf{if}\, (C[\sigma^{\uparrow}])\, (u[\sigma])\, (v[\sigma])\, (t[\sigma])$

Figure 1 Type theory as a generalised algebraic structure. $\sigma^{\uparrow}$ abbreviates $(\sigma \circ \mathsf{p}, \mathsf{q})$.

lifting of a substitution $\sigma : \mathsf{Sub}\,\Gamma\,\Delta$ by $\sigma^\uparrow : \mathsf{Sub}\,(\Gamma \triangleright A[\sigma])\,(\Delta \triangleright A) := (\sigma \circ \mathsf{p}, \mathsf{q})$. We observe that it has the property ${}^\uparrow[] : (\sigma^\uparrow)[\delta, t] = (\sigma \circ \delta, t)$.

Π-types are characterized by a natural isomorphism between $\mathsf{Tm}\,\Gamma\,(\Pi\,A\,B)$ and $\mathsf{Tm}\,(\Gamma \triangleright A)\,B$ (lam and app). We define the usual application as $t\,\$\,u := (\mathsf{app}\,t)[\mathsf{id}, u]$. $A \Rightarrow B$ abbreviates $\Pi\,A\,(B[\mathsf{p}])$. Σ-types are given by the constructor $-,-$ and projections projl and projr and they support an η-law. There is a unit type $\top$ with one constructor tt and an η-law and there is a hierarchy of Tarski-universes, given by natural isomorphisms between $\mathsf{Ty}\,i\,\Gamma$ and $\mathsf{Tm}\,\Gamma\,(\mathsf{U}\,i)$ for every i.[1] As terms of Π-, Σ- and U-types are characterized by natural isomorphisms, we stated the substitution law for only one of the two directions, the other can be derived. We illustrate how to do this for app and state the other laws.

$$
\begin{aligned}
\mathsf{app}[] &: (\mathsf{app}\,t)[\sigma^\uparrow] \overset{\Pi\beta}{=} \mathsf{app}\,(\mathsf{lam}\,((\mathsf{app}\,t)[\sigma^\uparrow])) \overset{\mathsf{lam}[]}{=} \mathsf{app}\,((\mathsf{lam}\,(\mathsf{app}\,t))[\sigma]) \overset{\Pi\eta}{=} \mathsf{app}\,(t[\sigma]) \\
\$[] &: (t\,\$\,u)[\sigma] = t[\sigma]\,\$\,u[\sigma] \\
\mathsf{projl}[] &: (\mathsf{projl}\,t)[\sigma] = \mathsf{projl}\,(t[\sigma]) \\
\mathsf{projr}[] &: (\mathsf{projr}\,t)[\sigma] = \mathsf{projr}\,(t[\sigma]) \\
\mathsf{c}[] &: (c\,A)[\sigma] = c\,(A[\sigma])
\end{aligned}
$$

Finally, we have booleans with a dependent eliminator if into any universe. Sometimes for readability we omit the first argument (C) of if and write _ instead.

Note that the well-typedness of some of the equations depends on previous equations. For example, the left-hand side of $\triangleright\beta_2$ has type $\mathsf{Tm}\,\Gamma\,(A[\mathsf{p}][\sigma, t])$, while the right-hand side has $\mathsf{Tm}\,\Gamma\,(A[\sigma])$, and these types are equal by $[\circ]$ and $\triangleright\beta_1$. In an intensional metatheory, we would need to transport the left-hand side over these equations. We use an extensional metatheory, so we do not write such dependencies.

As an example we write the polymorphic identity function as $\mathsf{lam}\,(\mathsf{lam}\,\mathsf{q})$. Note that lam and q have several implicit arguments that we did not write down. However, when we write a term, these implicit arguments should be clear from the context. In this example, saying that it has type $\mathsf{Tm}\,\cdot\,(\Pi\,(\mathsf{U}\,0)\,(\mathsf{El}\,\mathsf{q} \Rightarrow \mathsf{El}\,\mathsf{q}))$ fixes all its implicit arguments. We don't have raw terms with a type assignment or type inference system, we only work with fully annotated well-typed terms where lots of information is implicit (as usual in mathematics). This is sometimes called intrinsic or well-typed syntax, see [3] for an introduction.

We call algebras of the algebraic structure presented in Figure 1 *models* of type theory. When referring to different models, we put the model as a subscript, i.e. $\mathsf{Con}_\mathcal{M}$ refers to contexts in model $\mathcal{M}$, $\mathsf{id}_\mathcal{M} : \mathsf{Sub}_\mathcal{M}\,\Gamma_\mathcal{M}\,\Gamma_\mathcal{M}$ refers to the identity substitution in this model. For metavariables, we usually use the same subscript as for the two occurrences of $\Gamma_\mathcal{M}$ in the type of $\mathsf{id}_\mathcal{M}$.

We introduce some basic notions for working with models. These notions are obtained mechanically by viewing Figure 1 as a scheme for a quotient inductive-inductive type (QIIT, [18]).

[1] We learned this representation of universes from Thierry Coquand. Note that Russell universes could be represented by replacing El and c by the sort equation $\mathsf{Ty}\,i\,\Gamma = \mathsf{Tm}\,\Gamma\,(\mathsf{U}\,i)$ and the equation $A[\sigma]_{\mathsf{Ty}} = A[\sigma]_{\mathsf{Tm}}$ relating type and term substitutions. The latter equation is well-typed because of the former.

- A (strict) *morphism* H between models $\mathcal{M}$ and $\mathcal{N}$ consists of four functions between the sorts which preserve all the 26 operators (up to equality). We use subscripts to mark which component we mean, e.g. some of the components are the following.

$$
\begin{aligned}
&H_{\mathsf{Con}} : \mathsf{Con}_{\mathcal{M}}\, i \to \mathsf{Con}_{\mathcal{N}}\, i\\
&H_{\mathsf{Ty}} \;\;: \mathsf{Ty}_{\mathcal{M}}\, j\, \Gamma \to \mathsf{Ty}_{\mathcal{N}}\, j\,(H\,\Gamma)\\
&H_{\mathsf{Sub}} : \mathsf{Sub}_{\mathcal{M}}\, \Gamma\, \Delta \to \mathsf{Sub}_{\mathcal{N}}\,(H\,\Gamma)\,(H\,\Delta)\\
&H_{\mathsf{Tm}} \;: \mathsf{Tm}_{\mathcal{M}}\, \Gamma\, A \to \mathsf{Tm}_{\mathcal{N}}\,(H\,\Gamma)\,(H\,A)\\
&H_{[]} \quad\;: H_{\mathsf{Ty}}\,(A[\sigma]_{\mathcal{M}}) = (H_{\mathsf{Ty}}\,A)[H_{\mathsf{Sub}}\,\sigma]_{\mathcal{N}}\\
&H_{\triangleright} \quad\;: H_{\mathsf{Con}}\,(\Gamma \triangleright_{\mathcal{M}} A) = H_{\mathsf{Con}}\,\Gamma \triangleright_{\mathcal{N}} H_{\mathsf{Ty}}\,A\\
&H_{\Pi} \quad\;: H_{\mathsf{Ty}}\,(\Pi_{\mathcal{M}}\,A\,B) = \Pi_{\mathcal{N}}\,(H_{\mathsf{Ty}}\,A)\,(H_{\mathsf{Ty}}\,B)\\
&H_{\mathsf{lam}} \;: H_{\mathsf{Tm}}\,(\mathsf{lam}_{\mathcal{M}}\,t) = \mathsf{lam}_{\mathcal{N}}\,(H_{\mathsf{Tm}}\,t)\\
&H_{\mathsf{app}} \;: H_{\mathsf{Tm}}\,(\mathsf{app}_{\mathcal{M}}\,t) = \mathsf{app}_{\mathcal{N}}\,(H_{\mathsf{Tm}}\,t)
\end{aligned}
$$

Sometimes we omit subscripts for readability, e.g. above we wrote $H\,\Gamma$ instead of $H_{\mathsf{Con}}\,\Gamma$ and we also did not decorate metavariables with subscripts, all Γ above live in $\mathsf{Con}_{\mathcal{M}}$, all σ in $\mathsf{Sub}_{\mathcal{M}}$ etc. We will follow this convention later.

- A *displayed model* $\mathcal{Q}$ over a model $\mathcal{M}$ encodes a model with a strict morphism to $\mathcal{M}$. It is given by four families, 26 operations, and 34 equalities, all of which are over those of $\mathcal{M}$, e.g.

$$
\begin{aligned}
&\mathsf{Con}_{\mathcal{Q}} \;\;: (i : \mathbb{N}) \to \mathsf{Con}_{\mathcal{M}}\, i \to \mathsf{Set}\\
&\mathsf{Ty}_{\mathcal{Q}} \quad\;: (j : \mathbb{N}) \to \mathsf{Con}_{\mathcal{Q}}\, i\,\Gamma \to \mathsf{Ty}_{\mathcal{M}}\, j\,\Gamma \to \mathsf{Set}\\
&\mathsf{Sub}_{\mathcal{Q}} \;\;: \mathsf{Con}_{\mathcal{Q}}\, i\,\Gamma \to \mathsf{Con}_{\mathcal{Q}}\, j\,\Delta \to \mathsf{Sub}_{\mathcal{M}}\,\Gamma\,\Delta \to \mathsf{Set}\\
&\mathsf{Tm}_{\mathcal{Q}} \;\;\,: (\Gamma_{\mathcal{Q}} : \mathsf{Con}_{\mathcal{Q}}\, i\,\Gamma) \to \mathsf{Ty}_{\mathcal{Q}}\, j\,\Gamma_{\mathcal{Q}}\,A \to \mathsf{Tm}_{\mathcal{M}}\,\Gamma\,A \to \mathsf{Set}\\
&-[-]_{\mathcal{Q}} \;: \mathsf{Ty}_{\mathcal{Q}}\, j\,\Delta_{\mathcal{Q}}\,A \to \mathsf{Sub}_{\mathcal{Q}}\,\Gamma_{\mathcal{Q}}\,\Delta_{\mathcal{Q}}\,\sigma \to \mathsf{Ty}_{\mathcal{Q}}\, j\,\Gamma_{\mathcal{Q}}\,(A[\sigma]_{\mathcal{M}})\\
&- \triangleright_{\mathcal{Q}} - : (\Gamma_{\mathcal{Q}} : \mathsf{Con}_{\mathcal{Q}}\, i\,\Gamma) \to \mathsf{Ty}_{\mathcal{Q}}\, j\,\Gamma_{\mathcal{Q}}\,A \to \mathsf{Con}_{\mathcal{Q}}\,(i \sqcup j)\,(\Gamma \triangleright_{\mathcal{M}} A)\\
&\Pi_{\mathcal{Q}} \quad\;\;: (A_{\mathcal{Q}} : \mathsf{Ty}_{\mathcal{Q}}\, i\,\Gamma_{\mathcal{Q}}\,A) \to \mathsf{Ty}_{\mathcal{Q}}\, j\,(\Gamma_{\mathcal{Q}} \triangleright_{\mathcal{Q}} A_{\mathcal{Q}})\,B \to \mathsf{Ty}_{\mathcal{Q}}\,(i \sqcup j)\,\Gamma_{\mathcal{Q}}\,(\Pi_{\mathcal{M}}\,A\,B)\\
&\mathsf{lam}_{\mathcal{Q}} \;\;\,: \mathsf{Tm}_{\mathcal{Q}}\,(\Gamma_{\mathcal{Q}} \triangleright_{\mathcal{Q}} A_{\mathcal{Q}})\,B_{\mathcal{Q}}\,t \to \mathsf{Tm}_{\mathcal{Q}}\,\Gamma_{\mathcal{Q}}\,(\Pi_{\mathcal{Q}}\,A_{\mathcal{Q}}\,B_{\mathcal{Q}})\,(\mathsf{lam}_{\mathcal{M}}\,t)\\
&\mathsf{app}_{\mathcal{Q}} \;\;: \mathsf{Tm}_{\mathcal{Q}}\,\Gamma_{\mathcal{Q}}\,(\Pi_{\mathcal{Q}}\,A_{\mathcal{Q}}\,B_{\mathcal{Q}})\,t \to \mathsf{Tm}_{\mathcal{Q}}\,(\Gamma_{\mathcal{Q}} \triangleright_{\mathcal{Q}} A_{\mathcal{Q}})\,B_{\mathcal{Q}}\,(\mathsf{app}_{\mathcal{M}}\,t)\\
&\Pi\beta_{\mathcal{Q}} \;\;\,: \mathsf{app}_{\mathcal{Q}}\,(\mathsf{lam}_{\mathcal{Q}}\,t_{\mathcal{Q}}) = t_{\mathcal{Q}}
\end{aligned}
$$

- A *section* I of a displayed model $\mathcal{Q}$ over $\mathcal{M}$ is like a dependent morphism, encoding a section to the strict morphism to $\mathcal{M}$ encoded by $\mathcal{Q}$. It contains, among others, the following components.

$$
\begin{aligned}
&I_{\mathsf{Con}} \;: (\Gamma : \mathsf{Con}_{\mathcal{M}}\, i) \to \mathsf{Con}_{\mathcal{Q}}\, i\,\Gamma\\
&I_{\mathsf{Ty}} \;\;\,: (A : \mathsf{Ty}_{\mathcal{M}}\, j\,\Gamma) \to \mathsf{Ty}_{\mathcal{Q}}\, j\,(I\,\Gamma)\,A\\
&I_{\mathsf{Sub}} \;: (\sigma : \mathsf{Sub}_{\mathcal{M}}\,\Gamma\,\Delta) \to \mathsf{Sub}_{\mathcal{Q}}\,(I\,\Gamma)\,(I\,\Delta)\,\sigma\\
&I_{A[\sigma]} : I\,(A[\sigma]_{\mathcal{M}}) = (I\,A)[I\,\sigma]_{\mathcal{Q}}\\
&I_{\triangleright} \quad\;: I\,(\Gamma \triangleright_{\mathcal{M}} A) = I\,\Gamma \triangleright_{\mathcal{Q}} I\,A\\
&I_{\Pi} \quad\;: I\,(\Pi_{\mathcal{M}}\,A\,B) = \Pi_{\mathcal{Q}}\,(I\,A)\,(I\,B)\\
&I_{\mathsf{lam}} \;\;: I\,(\mathsf{lam}_{\mathcal{M}}\,t) = \mathsf{lam}_{\mathcal{Q}}\,(I\,t)\\
&I_{\mathsf{app}} \;\;: I\,(\mathsf{app}_{\mathcal{M}}\,t) = \mathsf{app}_{\mathcal{Q}}\,(I\,t)
\end{aligned}
$$

We assume the existence of the quotient inductive-inductive type (QIIT, [18]) specified by Figure 1. We thus have an initial model S, called the *syntax*. For every model $\mathcal{M}$, the *recursor* $\mathsf{rec}^{\mathcal{M}}$ is the unique morphism from S to $\mathcal{M}$. For every displayed model $\mathcal{Q}$ over S, the *eliminator* $\mathsf{elim}^{\mathcal{Q}}$ is the unique section of $\mathcal{Q}$.

2.1 The identity type

In our construction of gluing we will assume that the target model has identity types. Identity types extend type theory as given in Figure 1 with the following operators and equations.

$$
\begin{aligned}
&\mathsf{Id} &&: (A : \mathsf{Ty}\,i\,\Gamma) \to \mathsf{Tm}\,\Gamma\,A \to \mathsf{Tm}\,\Gamma\,A \to \mathsf{Ty}\,i\,\Gamma \\
&\mathsf{refl} &&: (u : \mathsf{Tm}\,\Gamma\,A) \to \mathsf{Tm}\,\Gamma\,(\mathsf{Id}\,A\,u\,u) \\
&\mathsf{J} &&: \big(C : \mathsf{Ty}\,i\,(\Gamma \triangleright A \triangleright \mathsf{Id}\,(A[\mathsf{p}])\,(u[\mathsf{p}])\,0)\big) \to \mathsf{Tm}\,\Gamma\,(C[\mathsf{id}, u, \mathsf{refl}\,u]) \to \\
& &&\quad (e : \mathsf{Tm}\,\Gamma\,(\mathsf{Id}\,A\,u\,v)) \to \mathsf{Tm}\,\Gamma\,(C[\mathsf{id}, v, e[\mathsf{p}]]) \\
&\mathsf{Id}\beta &&: \mathsf{J}\,C\,w\,(\mathsf{refl}\,u) = w \\
&\mathsf{Id}[] &&: (\mathsf{Id}\,A\,u\,v)[\sigma] = \mathsf{Id}\,(A[\sigma])\,(u[\sigma])\,(v[\sigma]) \\
&\mathsf{refl}[] &&: (\mathsf{refl}\,u)[\sigma] = \mathsf{refl}\,(u[\sigma]) \\
&\mathsf{J}[] &&: (\mathsf{J}\,C\,w\,e)[\sigma] = \mathsf{J}\,(C[\sigma^{\uparrow\uparrow}])\,(w[\sigma])\,(e[\sigma])
\end{aligned}
$$

$\mathsf{Id}\,A\,u\,v$ expresses that u is equal to v, there is one constructor refl expressing reflexivity and there is the eliminator J which says that given a family over identities and a witness of that family for refl we get that there is an element of that family for every identity proof.

3 The Set model

As an example of a simple model, we define the set model (standard model, metacircular model). In this model, contexts are sets, types are families over their contexts, substitutions are functions, and terms are dependent functions. Context extension is metatheoretic Σ, otherwise everything is modelled by its metatheoretic counterparts, e.g. Π-types are dependent functions, lam is λ, app is metatheoretic application. We list a few components for illustration.

$$
\begin{aligned}
&\mathsf{Con}\,i &&:= \mathsf{Set}_i \\
&\mathsf{Ty}\,j\,\Gamma &&:= \Gamma \to \mathsf{Set}_j \\
&\mathsf{Sub}\,\Gamma\,\Delta &&:= \Gamma \to \Delta \\
&\mathsf{Tm}\,\Gamma\,A &&:= (\gamma : \Gamma) \to A\,\gamma \\
&A[\sigma] &&:= \lambda\gamma.A\,(\sigma\,\gamma) \\
&\cdot &&:= \mathbb{1} \\
&\epsilon &&:= \lambda_.* \\
&\Gamma \triangleright A &&:= (\gamma : \Gamma) \times A\,\gamma \\
&(\sigma, t) &&:= (\sigma, t) \\
&\mathsf{p} &&:= \mathsf{projl} \\
&\mathsf{q} &&:= \mathsf{projr} \\
&\Pi\,A\,B &&:= \lambda\gamma.(\alpha : A\,\gamma) \to B\,(\gamma, \alpha)
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{lam}\, t && := \lambda\gamma.\lambda\alpha.t\,(\gamma,\alpha)\\
&\mathsf{app}\, t && := \lambda\gamma.t\,\gamma_{.1}\,\gamma_{.2}\\
&\Pi\beta && : \mathsf{app}\,(\mathsf{lam}\, t) = \lambda\gamma'.(\lambda\gamma.\lambda\alpha.t\,(\gamma,\alpha))\,\gamma'_{.1}\,\gamma'_{.2} \overset{\rightarrow\beta}{=} \lambda\gamma'.t\,(\gamma'_{.1},\gamma'_{.2}) \overset{\times\eta}{=} \lambda\gamma'.t\,\gamma' \overset{\rightarrow\eta}{=} t\\
&\mathsf{U}\, i && := \lambda_.\mathsf{Set}_i\\
&\mathsf{El}\, a && := a\\
&\mathsf{c}\, a && := a\\
&\mathsf{Bool} && := \mathbb{2}\\
&\mathsf{true} && := *\\
&\mathsf{false} && := **\\
&\mathsf{if}\, C\, t\, u\, v && := \mathsf{case}\, t\, u\, v\\
&\mathsf{Id}\, A\, u\, v && := (u = v)
\end{aligned}
$$

The β-law for Π uses the metatheoretic β- and η-laws for the functions and η for pairs.

Using the recursor we can define an interpreter for our syntax which maps syntactic terms to metatheoretic objects.

$$
\begin{aligned}
&[\![-]\!] : \mathsf{Con}_{\mathsf{S}}\, i && \to \mathsf{Set}_i && := \mathsf{rec}^{\mathsf{Set}}_{\mathsf{Con}}\\
&[\![-]\!] : \mathsf{Ty}_{\mathsf{S}}\, j\, \Gamma && \to [\![\Gamma]\!] \to \mathsf{Set}_j && := \mathsf{rec}^{\mathsf{Set}}_{\mathsf{Ty}}\\
&[\![-]\!] : \mathsf{Sub}_{\mathsf{S}}\, \Gamma\, \Delta && \to [\![\Gamma]\!] \to [\![\Delta]\!] && := \mathsf{rec}^{\mathsf{Set}}_{\mathsf{Sub}}\\
&[\![-]\!] : \mathsf{Tm}_{\mathsf{S}}\, \Gamma\, A && \to (\gamma : [\![\Gamma]\!]) \to [\![A]\!]\,\gamma && := \mathsf{rec}^{\mathsf{Set}}_{\mathsf{Tm}}
\end{aligned}
$$

For example, the interpretation of the polymorphic identity function is

$$[\![\mathsf{lam}\,(\mathsf{lam}\,\mathsf{q})]\!] : (\gamma : \mathbb{1}) \to (A : \mathsf{Set}_0) \to A \to A = \lambda\gamma.\lambda A.\lambda a.a.$$

4 Pseudomorphism

In this section we define morphisms of models of type theory which are strict on the category structure and weak on $\cdot$ and $-\triangleright-$. We call such a morphism a *pseudomorphism*, its components are listed in Figure 2. In terms of using comprehension categories [17] to describe models of type theory, this corresponds to the notion of morphism that relates the Grothendieck fibrations strictly, but the comprehension maps only up to natural isomorphism.

Just as a strict morphism (described in Section 2), a pseudomorphism F maps contexts in $\mathcal{S}$ to contexts in $\mathcal{M}$, types in $\mathcal{S}$ to types in $\mathcal{M}$, etc. Identity, composition and action on substitution are preserved strictly (F_{id}, $F_{\circ}$, $F_{[]}$ and $F_{[]}$). The empty context and context extension are preserved up to definitional isomorphism. Definitional isomorphism between two contexts $\Gamma : \mathsf{Con}\, i$, $\Delta : \mathsf{Con}\, j$ is defined as follows.

$$(f : \Gamma \cong \Delta) := (f_{.1} : \mathsf{Sub}\,\Gamma\,\Delta) \times (f_{.2} : \mathsf{Sub}\,\Delta\,\Gamma) \times (f_{.12} : f_{.1} \circ f_{.2} = \mathsf{id}) \times (f_{.21} : f_{.2} \circ f_{.1} = \mathsf{id})$$

$F_{\triangleright.1\circ}$ denotes a naturality condition that $F_{\triangleright}$ has to satisfy. The empty substitution ϵ and the comprehension operators $(-,-)$, p, q are preserved strictly, but this is up to the weakness of $\cdot$ and $\triangleright$.

Categorically, a pseudomorphism is a functor on categories of contexts with natural transformations on types and terms with the following properties: it preserves terminal objects; given $A : \mathsf{Ty}_{\mathcal{S}}\, j\, \Delta$, if a pair of $\sigma : \mathsf{Sub}_{\mathcal{S}}\,\Gamma\,\Delta$ and $t : \mathsf{Tm}_{\mathcal{S}}\,\Gamma\,(A[\sigma]_{\mathcal{S}})$ has the universal property of the context extension of Δ with A in $\mathcal{S}$, then the pair of $F\,\sigma$ and $F\,t$ has the universal property of the context extension of $F\,\Delta$ with $F\,A$ in $\mathcal{M}$.

$$\begin{aligned}
F_{\mathsf{Con}} &: \mathsf{Con}_{\mathcal{S}}\, i \to \mathsf{Con}\, i \\
F_{\mathsf{Ty}} &: \mathsf{Ty}_{\mathcal{S}}\, j\, \Gamma \to \mathsf{Ty}\, j\, (F\, \Gamma) \\
F_{\mathsf{Sub}} &: \mathsf{Sub}_{\mathcal{S}}\, \Gamma\, \Delta \to \mathsf{Sub}\, (F\, \Gamma)\, (F\, \Delta) \\
F_{\mathsf{Tm}} &: \mathsf{Tm}_{\mathcal{S}}\, \Gamma\, A \to \mathsf{Tm}\, (F\, \Gamma)\, (F\, A) \\
F_{\mathsf{id}} &: F\, \mathsf{id}_{\mathcal{S}} = \mathsf{id} \\
F_{\circ} &: F\, (\sigma \circ_{\mathcal{S}} \delta) = F\, \sigma \circ F\, \delta \\
F_{[]} &: F\, (A[\sigma]_{\mathcal{S}}) = (F\, A)[F\, \sigma] \\
F_{[]} &: F\, (t[\sigma]_{\mathcal{S}}) = (F\, t)[F\, \sigma] \\
F_{\cdot} &: F\, \cdot_{\mathcal{S}} \cong \cdot \\
F_{\epsilon} &: F\, \epsilon_{\mathcal{S}} = F_{\cdot.2} \circ \epsilon \\
F_{\triangleright} &: F\, (\Gamma \triangleright_{\mathcal{S}} A) \cong F\, \Gamma \triangleright F\, A \\
F_{\triangleright.1\circ} &: F_{\triangleright.1} \circ F\, (\sigma^{\uparrow_{\mathcal{S}}}) = (F\, \sigma)^{\uparrow} \circ F_{\triangleright.1} \\
F_{,} &: F\, (\sigma ,_{\mathcal{S}} t) = F_{\triangleright.2} \circ (F\, \sigma, F\, t) \\
F_{\mathsf{p}} &: F\, \mathsf{p}_{\mathcal{S}} = \mathsf{p} \circ F_{\triangleright.1} \\
F_{\mathsf{q}} &: F\, \mathsf{q}_{\mathcal{S}} = \mathsf{q}[F_{\triangleright.1}]
\end{aligned}$$

Figure 2 The components of a pseudomorphism F from $\mathcal{S}$ to $\mathcal{M}$. For readability, we omit the subscripts $_{\mathcal{S}}$ from the metavariable names and all the $_{\mathcal{M}}$ subscripts. That is, when we write Con or id we mean $\mathsf{Con}_{\mathcal{M}}$ and $\mathsf{id}_{\mathcal{M}}$. We overload the different parts of F, i.e. write F for F_{Con}, F_{Ty}, F_{Sub} and F_{Tm}. $\cong$ denotes definitional isomorphism, see in the text.

We derive the following naturality condition.

$$\begin{aligned}
F_{\triangleright.2\circ} : F_{\triangleright.2} \circ_{\mathcal{M}} (F\, \sigma)^{\uparrow_{\mathcal{M}}} &\overset{F_{\triangleright.12}}{=} F_{\triangleright.2} \circ (F\, \sigma)^{\uparrow_{\mathcal{M}}} \circ F_{\triangleright.1} \circ F_{\triangleright.2} \overset{F_{\triangleright.1\circ}}{=} \\
& F_{\triangleright.2} \circ F_{\triangleright.1} \circ F\, (\sigma^{\uparrow_{\mathcal{S}}}) \circ F_{\triangleright.2} \overset{F_{\triangleright.21}}{=} F\, (\sigma^{\uparrow_{\mathcal{S}}}) \circ_{\mathcal{M}} F_{\triangleright.2}
\end{aligned}$$

From this it follows that $F\, (\sigma^{\uparrow_{\mathcal{S}}}) = F_{\triangleright.2} \circ_{\mathcal{M}} (F\, \sigma)^{\uparrow_{\mathcal{M}}} \circ_{\mathcal{M}} F_{\triangleright.1}$.

Note that every strict morphism F is automatically pseudo, with $F_{\cdot.1} = F_{\cdot.2} = \mathsf{id}_{\mathcal{M}}$ and $F_{\triangleright.1} = F_{\triangleright.2} = \mathsf{id}_{\mathcal{M}}$.

A pseudomorphism automatically preserves type formers: Σ and $\top$ are preserved weakly, Π and U are preserved in a lax way and Bool in an oplax way. For example, we can define a map from $F\, (\Pi\, A\, B)$ to $\Pi\, (F\, A)\, (F\, B[F_{\triangleright.2}])$ as follows. We start using the eliminator of Π in $\mathcal{S}$ on a variable:

$$\mathsf{app}\, \mathsf{q} : \mathsf{Tm}_{\mathcal{S}}\, (\Gamma \triangleright \Pi\, A\, B \triangleright A[\mathsf{p}])\, (B[\mathsf{p}^{\uparrow}]).$$

Then we apply the pseudomorphism F and get a map in $\mathcal{M}$:

$$F\, (\mathsf{app}\, \mathsf{q}) : \mathsf{Tm}_{\mathcal{M}}\, \big(F\, (\Gamma \triangleright \Pi\, A\, B \triangleright A[\mathsf{p}])\big)\, \big(F\, (B[\mathsf{p}^{\uparrow}])\big).$$

Applying the substitution $F_{\triangleright.2} \circ (F_{\triangleright.2}{}^{\uparrow})$ and using the properties of F we obtain

$$F\, (\mathsf{app}\, \mathsf{q})[F_{\triangleright.2} \circ (F_{\triangleright.2}{}^{\uparrow})] : \mathsf{Tm}_{\mathcal{M}}\, \big(F\, \Gamma \triangleright F\, (\Pi\, A\, B) \triangleright F\, A[\mathsf{p}]\big)\, \big(F\, B[F_{\triangleright.2} \circ (\mathsf{p}^{\uparrow})]\big),$$

and finally we use the constructor of Π in $\mathcal{M}$:

$$\mathsf{lam}\, \big(F\, (\mathsf{app}\, \mathsf{q})[F_{\triangleright.2} \circ (F_{\triangleright.2}{}^{\uparrow})]\big) : \mathsf{Tm}_{\mathcal{M}}\, \big(F\, \Gamma \triangleright F\, (\Pi\, A\, B)\big)\, \big(\Pi\, (F\, A)\, (F\, B[F_{\triangleright.2}])\big).$$

We define a similar map for U, a map from Bool to $F\,\mathsf{Bool}$ and maps in both directions for Σ and $\top$. To express the latter we introduce the following abbreviation for a definitional isomorphism between types $A, B : \mathsf{Ty}\,\Gamma$.

$$(g : A \cong_\Gamma B) := (g_{.1} : \mathsf{Tm}\,(\Gamma \rhd A)\,(B[\mathsf{p}])) \times (g_{.2} : \mathsf{Tm}\,(\Gamma \rhd B)\,(A[\mathsf{p}]))\times$$
$$(g_{.12} : g_{.1}[\mathsf{p}, g_{.2}] = \mathsf{q}) \times (g_{.21} : g_{.2}[\mathsf{p}, g_{.1}] = \mathsf{q})$$

For a pseudomorphism F the following comparison maps can be given.

$$\begin{aligned}
F_{\Pi.1} &: \mathsf{Tm}\,\big(F\,\Gamma \rhd F\,(\Pi\,A\,B)\big)\,\big(\Pi\,(F\,A)\,(F\,B[F_{\rhd.2}])\,[\mathsf{p}]\big)\\
F_{\mathsf{U}.1} &: \mathsf{Tm}\,(F\,\Gamma \rhd F\,(\mathsf{U}\,i))\,(\mathsf{U}\,i)\\
F_{\mathsf{Bool}.2} &: \mathsf{Tm}\,(F\,\Gamma \rhd \mathsf{Bool})\,(F\,\mathsf{Bool}[\mathsf{p}])\\
F_{\Sigma} &: F\,(\Sigma_{\mathcal{S}}\,A\,B) \cong_{F\,\Gamma} \Sigma_{\mathcal{M}}\,(F\,A)\,(F\,B[F_{\rhd.2}])\\
F_{\top} &: F\,\top \cong_{F\,\Gamma} \top
\end{aligned}$$

The definition of $F_{\mathsf{U}.1}$ is similar to that of $F_{\Pi.1}$: we first use the eliminator El on a variable, then apply F, then adjust the context using $F_{\rhd.2}$, finally use the constructor c. For Bool, we go in the other direction and use the eliminator in $\mathcal{M}$ on a variable and return the corresponding constructors in $\mathcal{S}$ with F applied to them. For Σ we combine these methods to obtain maps in both directions. The maps for $\top$ are trivial.

$$\begin{aligned}
F_{\Pi.1} &:= \mathsf{lam}\,\big(F\,(\mathsf{app}\,\mathsf{q})[F_{\rhd.2} \circ (F_{\rhd.2}{}^{\uparrow})]\big)\\
F_{\mathsf{U}.1} &:= \mathsf{c}\,\big(F\,(\mathsf{El}\,\mathsf{q})[F_{\rhd.2}]\big)\\
F_{\mathsf{Bool}.2} &:= \mathsf{if_q}\,(F\,\mathsf{true}[\mathsf{p}])\,(F\,\mathsf{false}[\mathsf{p}])\\
F_{\Sigma.1} &:= \big(F\,(\mathsf{projl}\,\mathsf{q}), F\,(\mathsf{projr}\,\mathsf{q})\big)[F_{\rhd.2}]\\
F_{\Sigma.2} &:= F\,(1,0)\big[F_{\rhd.2} \circ (F_{\rhd.2} \circ (\mathsf{p}, \mathsf{projl}\,\mathsf{q}), \mathsf{projr}\,\mathsf{q})\big]\\
F_{\Sigma.12} &:\quad F_{\Sigma.1}[\mathsf{p}, F_{\Sigma.2}] =\\
&\quad\ (F\,(\mathsf{projl}\,\mathsf{q}), F\,(\mathsf{projr}\,\mathsf{q}))[F\,(\mathsf{p}, (1,0))][F_{\rhd.2} \circ (F_{\rhd.2} \circ (\mathsf{p}, \mathsf{projl}\,\mathsf{q}), \mathsf{projr}\,\mathsf{q})] =\\
&\quad\ (\mathsf{projl}\,\mathsf{q}, \mathsf{projr}\,\mathsf{q}) \overset{\Sigma\eta_{\mathcal{M}}}{=} \mathsf{q}\\
F_{\Sigma.21} &:\quad F_{\Sigma.2}[\mathsf{p}, F_{\Sigma.1}] =\\
&\quad\ F\,(1,0)[F\,(\mathsf{p}, \mathsf{projl}\mathsf{q}, \mathsf{projr}\,q) \circ F_{\rhd.2}] =\\
&\quad\ F\,(\mathsf{projl}\,\mathsf{q}, \mathsf{projr}\,q)[F_{\rhd.2}] \overset{\Sigma\eta_{\mathcal{S}}}{=} F\,\mathsf{q}[F_{\rhd.2}] \overset{F_{\mathsf{q}}, F_{\rhd}}{=} \mathsf{q}\\
F_{\top.1} &:= \mathsf{tt}\\
F_{\top.2} &:= F\,\mathsf{tt}[\mathsf{p}]\\
F_{\top.12} &:\quad F_{\top.1}[\mathsf{p}, F_{\top.2}] = \mathsf{tt}[\mathsf{p}, F\,\mathsf{tt}[\mathsf{p}]] = \mathsf{tt} \overset{\top\eta_{\mathcal{M}}}{=} \mathsf{q}\\
F_{\top.21} &:\quad F_{\top.2}[\mathsf{p}, F_{\top.1}] = F\,\mathsf{tt}[\mathsf{p}] = F\,(\mathsf{tt}[\mathsf{p}])[F_{\rhd.2}] \overset{\top\eta_{\mathcal{S}}}{=} F\,\mathsf{q}[F_{\rhd.2}] = \mathsf{q}
\end{aligned}$$

At the end of Section 6 we remark on the (im)possibility of comparison maps in the other direction such as $F_{\Pi.2}$.

5 Gluing

In this section, given a pseudomorphism F from model $\mathcal{S}$ to model $\mathcal{M}$, we define a displayed model P^F (P for short) over $\mathcal{S}$. We call this model *gluing* along F and its components are given in Figure 3. We omit some $_{\mathcal{S}}$ and all $_{\mathcal{M}}$ subscripts for readability.

$$
\begin{aligned}
&\mathsf{Con}_{\mathsf{P}}\, i\, \Gamma && := \mathsf{Ty}\, i\, (F\, \Gamma)\\
&\mathsf{Ty}_{\mathsf{P}}\, j\, \Gamma_{\mathsf{P}}\, A && := \mathsf{Ty}\, j\, (F\, \Gamma \triangleright \Gamma_{\mathsf{P}} \triangleright F\, A[\mathsf{p}])\\
&\mathsf{Sub}_{\mathsf{P}}\, \Gamma_{\mathsf{P}}\, \Delta_{\mathsf{P}}\, \sigma && := \mathsf{Tm}\, (F\, \Gamma \triangleright \Gamma_{\mathsf{P}})\, (\Delta_{\mathsf{P}}[F\, \sigma \circ \mathsf{p}])\\
&\mathsf{Tm}_{\mathsf{P}}\, \Gamma_{\mathsf{P}}\, A_{\mathsf{P}}\, t && := \mathsf{Tm}\, (F\, \Gamma \triangleright \Gamma_{\mathsf{P}})\, (A_{\mathsf{P}}[\mathsf{id}, F\, t[\mathsf{p}]])\\
&\mathsf{id}_{\mathsf{P}} && := \mathsf{q}\\
&\sigma_{\mathsf{P}} \circ_{\mathsf{P}} \delta_{\mathsf{P}} && := \sigma_{\mathsf{P}}[F\, \delta \circ \mathsf{p}, \delta_{\mathsf{P}}]\\
&A_{\mathsf{P}}[\sigma_{\mathsf{P}}]_{\mathsf{P}} && := A_{\mathsf{P}}[F\, \sigma \circ \mathsf{p}^2, \sigma_{\mathsf{P}}[\mathsf{p}], \mathsf{q}]\\
&t_{\mathsf{P}}[\sigma_{\mathsf{P}}]_{\mathsf{P}} && := t_{\mathsf{P}}[F\, \sigma \circ \mathsf{p}, \sigma_{\mathsf{P}}]\\
&\cdot_{\mathsf{P}} && := \top\\
&\epsilon_{\mathsf{P}} && := \mathsf{tt}\\
&\Gamma_{\mathsf{P}} \triangleright_{\mathsf{P}} A_{\mathsf{P}} && := \Sigma\, \big(\Gamma_{\mathsf{P}}[\mathsf{p} \circ F_{\triangleright.1}]\big)\, \big(A_{\mathsf{P}}[\mathsf{p} \circ F_{\triangleright.1} \circ \mathsf{p}, 0, \mathsf{q}[F_{\triangleright.1} \circ \mathsf{p}]]\big)\\
&\sigma_{\mathsf{P}} ,_{\mathsf{P}} t_{\mathsf{P}} && := (\sigma_{\mathsf{P}}, t_{\mathsf{P}})\\
&\mathsf{p}_{\mathsf{P}} && := \mathsf{projl}\, \mathsf{q}\\
&\mathsf{q}_{\mathsf{P}} && := \mathsf{projr}\, \mathsf{q}\\
&\Pi_{\mathsf{P}}\, A_{\mathsf{P}}\, B_{\mathsf{P}} && := \Pi\, \big(F\, A[\mathsf{p}^2]\big)\, \Big(\Pi\, \big(A_{\mathsf{P}}[\mathsf{p}^2, \mathsf{q}]\big)\, \big(B_{\mathsf{P}}\big[F_{\triangleright.2} \circ (\mathsf{p}^4, 1), (3, 0), F_{\Pi.1}[\mathsf{p}^4, 2]\,\$\,1\big]\big)\Big)\\
&\mathsf{lam}\, t_{\mathsf{P}} && := \mathsf{lam}\, \Big(\mathsf{lam}\, \big(t_{\mathsf{P}}\big[F_{\triangleright.2} \circ (\mathsf{p}^3, 1), (2, 0)\big]\big)\Big)\\
&\mathsf{app}\, t_{\mathsf{P}} && := \big(\mathsf{app}\, (\mathsf{app}\, t_{\mathsf{P}})\big)\big[\mathsf{p} \circ F_{\triangleright.1} \circ \mathsf{p}, \mathsf{projl}\, 0, 0[F_{\triangleright.1} \circ \mathsf{p}], \mathsf{projr}\, 0\big]\\
&\Sigma_{\mathsf{P}}\, A_{\mathsf{P}}\, B_{\mathsf{P}} && := \Sigma\, \big(A_{\mathsf{P}}\big[\mathsf{p}, \mathsf{projl}\, (F_{\Sigma.1}[\mathsf{p}^2, \mathsf{q}])\big]\big)\\
& && \qquad \big(B_{\mathsf{P}}\big[F_{\triangleright.2} \circ (\mathsf{p}^3, \mathsf{projl}\, (F_{\Sigma.1}[\mathsf{p}^2, \mathsf{q}])), (2, 0), \mathsf{projr}\, (F_{\Sigma.1}[\mathsf{p}^2, \mathsf{q}])\big]\big)\\
&(u_{\mathsf{P}} ,_{\mathsf{P}} v_{\mathsf{P}}) && := (u_{\mathsf{P}}, v_{\mathsf{P}})\\
&\mathsf{projl}_{\mathsf{P}}\, t_{\mathsf{P}} && := \mathsf{projl}\, t_{\mathsf{P}}\\
&\mathsf{projr}_{\mathsf{P}}\, t_{\mathsf{P}} && := \mathsf{projr}\, t_{\mathsf{P}}\\
&\top_{\mathsf{P}} && := \top\\
&\mathsf{tt}_{\mathsf{P}} && := \mathsf{tt}\\
&\mathsf{U}_{\mathsf{P}}\, i && := \mathsf{El}\, (F_{\mathsf{U}.1}[\mathsf{p}^2, \mathsf{q}]) \Rightarrow \mathsf{U}\, i\\
&\mathsf{El}_{\mathsf{P}}\, a_{\mathsf{P}} && := \mathsf{El}\, (\mathsf{app}\, a_{\mathsf{P}})\\
&\mathsf{c}_{\mathsf{P}}\, A_{\mathsf{P}} && := \mathsf{lam}\, (\mathsf{c}\, A_{\mathsf{P}})\\
&\mathsf{Bool}_{\mathsf{P}} && := \Sigma\, \mathsf{Bool}\, \big(\mathsf{Id}\, (F\, \mathsf{Bool}_{\mathcal{S}}[\mathsf{p}^3])\, (F_{\mathsf{Bool}.2}[\mathsf{p}^3, \mathsf{q}])\, 1\big)\\
&\mathsf{true}_{\mathsf{P}} && := (\mathsf{true}, \mathsf{refl}\, (F\, \mathsf{true}_{\mathcal{S}}[\mathsf{p}]))\\
&\mathsf{false}_{\mathsf{P}} && := (\mathsf{false}, \mathsf{refl}\, (F\, \mathsf{false}_{\mathcal{S}}[\mathsf{p}]))\\
&\mathsf{if}_{\mathsf{P}}\, C_{\mathsf{P}}\, t_{\mathsf{P}}\, u_{\mathsf{P}}\, v_{\mathsf{P}} && := \mathsf{J}\, _\, (\mathsf{if}\, _\, (\mathsf{projl}\, t_{\mathsf{P}})\, u_{\mathsf{P}}\, v_{\mathsf{P}})\, (\mathsf{projr}\, t_{\mathsf{P}})
\end{aligned}
$$

Figure 3 The displayed model P^F obtained by gluing along F. We write P instead of P^F, we omit some $_{\mathcal{S}}$ and all $_{\mathcal{M}}$ subscripts for readability. The full version of if_{P} (with the _s filled in) is given in Appendix B.

In the introduction we remarked that in categorical gluing an object in the glued model consists of a triple $\Gamma : |\mathcal{S}|$, $\Delta : |\mathcal{M}|$ and a morphism $\mathcal{M}(\Delta, F\,\Gamma)$. We could follow this line and define the gluing as a model with contexts such triples that comes with a strict "projection" morphism to $\mathcal{S}$. This could be called the fibrational or display map approach. Instead our definition is more type theoretic, it uses indexed families, doubly (for the correspondence between fibrations and families see e.g. [8, p. 221]). Firstly, the glued model is given as a displayed model, that is, for each $\Gamma : \mathsf{Con}_{\mathcal{S}}\,i$ we have a set $\mathsf{Con}_{\mathsf{P}}\,i\,\Gamma$. Secondly, instead of setting $\mathsf{Con}_{\mathsf{P}}\,i\,\Gamma$ to $(\Delta : \mathsf{Con}_{\mathcal{M}}\,i) \times \mathsf{Sub}_{\mathcal{M}}\,\Delta\,(F\,\Gamma)$, we use the built-in notion of indexed families in $\mathcal{M}$, that is: types. Hence a context over Γ is an $\mathcal{M}$-type in context $F\,\Gamma$. We remark that the glueing construction also works with the former choice of contexts.

Types in type theory can be thought of as proof-relevant predicates over their context and this is the intuition we adopt for describing the glued model. This is in line with the logical predicate view of gluing. We start with $\mathsf{Con}_{\mathsf{P}}\,i\,\Gamma$: a predicate at Γ is indexed over the F-image of Γ. A predicate at a type A is indexed over the image of Γ for which the predicate holds and the image of A. For a substitution $\sigma : \mathsf{Sub}\,\Gamma\,\Delta$, we state the fundamental lemma: if the predicate holds at Γ, the predicate holds at Δ for the F-image of the substitution. In short, images of substitutions respect the predicate. For terms, we similarly state that the image of a term respects the predicate.

We continue by explaining what the logical predicate says at different contexts and types. The predicate at the empty context $\cdot_{\mathsf{P}}$ is always true. At extended contexts the predicate is given pointwise by a Σ-type. $\Gamma_{\mathsf{P}} \triangleright A_{\mathsf{P}}$ is in context $F\,(\Gamma \triangleright A)$, but Γ_{P} only needs the component $F\,\Gamma$, which we obtain using the isomorphism $F_{\triangleright.1}$ from $F\,(\Gamma \triangleright_{\mathcal{S}} A)$ to $F\,\Gamma \triangleright F\,A$ followed by first projection. A_{P} is first indexed over $F\,\Gamma$, which is given by $\mathsf{p} \circ F_{\triangleright.1} \circ \mathsf{p}$, then over Γ_{P}, which is the first component of the Σ-type referenced by q, then over $F\,A$, which is provided by the $F_{\triangleright.1}$ part of the isomorphism.

The predicate at a Π-type holds for a function of type $F\,(\Pi\,A\,B)$ if whenever it holds for an input, it holds for the output. Let's look at how we express that the predicate holds at B for the output! We are in context

$$\Theta := F\,\Gamma \triangleright \underbrace{\Gamma_{\mathsf{P}}}_{3} \triangleright \underbrace{F\,(\Pi_{\mathcal{S}}\,A\,B)}_{2} \triangleright \underbrace{F\,A[\mathsf{p}^2]}_{1} \triangleright \underbrace{A_{\mathsf{P}}[\mathsf{p}^2, \mathsf{q}]}_{0}$$

where we wrote the de Bruijn indices referring to each component underneath. B_{P} is a predicate indexed over $F\,(\Gamma \triangleright_{\mathcal{S}} A)$, $\Gamma_{\mathsf{P}} \triangleright_{\mathsf{P}} A_{\mathsf{P}}$ and $F\,B[\mathsf{p}]$. The first index is given by $F_{\triangleright.2}$, which puts together the $F\,\Gamma$ (forgetting the last four elements in Θ by p^4) and the $F\,A$ components (last but one element in Θ, i.e. 1). The second index is given by de Bruijn indices 3 and 0. The last index is the result of applying the function given by De Bruijn index 2. We have to use the comparison map $F_{\Pi.1}$ defined in Section 4 which turns an $F\,(\Pi\,A\,B)$ into a $\Pi\,(F\,A)\,(F\,B[F_{\triangleright.2}])$. We supply its dependencies $F\,\Gamma$ by p^4 and $F\,(\Pi\,A\,B)$ by 2 and we use old-style application \$ with input 1 to get the result.

The predicate at a Σ-type holds if it holds pointwise. Here we use the comparison map $F_{\Sigma.1}$ combined with projl and projr to obtain $F\,A$ and $F\,B$ from $F\,(\Sigma_{\mathcal{S}}\,A\,B)$. The predicate at $\top$ is trivial. The predicate at the universe is the space of predicates expressed as functions into $\mathsf{U}\,i$. The domain of this function space is again obtained by applying the comparison map $F_{\mathsf{U}.1}$. The predicate at Bool for b in $F\,\mathsf{Bool}$ says that there is an $\mathcal{M}$-boolean to which we apply the comparison map $F_{\mathsf{Bool}.2}$ (which turns it into $F\,\mathsf{Bool}$), the result is equal to b.

The substition and term part of the gluing model is fairly straightforward. The most interesting component is if_{P} where we use J to eliminate the right projection of t_{P} (which is the equality in the second component of $\mathsf{Bool}_{\mathsf{P}}$), then we case split on the first projection

by $\mathsf{if}_{\mathcal{M}}$ and return u_{P} and v_{P} in the true and false cases, respectively. We omitted some arguments of J and if for readability, the full version is given in Appendix B. There we also verify that all equalities of the displayed model of type theory hold.

6 Global section functor

In this section we define the global section functor and show that it is a pseudomorphism. In the next section we will use this property to derive canonicity for type theory.

A model $\mathcal{S}$ *supports a global section functor* if it has the following two properties:

- $\mathsf{Sub}_{\mathcal{S}}\ \cdot_{\mathcal{S}}\ \Gamma : \mathsf{Set}_i$ whenever $\Gamma : \mathsf{Con}_{\mathcal{S}}\ i$
- $\mathsf{Tm}_{\mathcal{S}}\ \cdot_{\mathcal{S}}\ (A[\rho]_{\mathcal{S}}) : \mathsf{Set}_j$ whenever $A : \mathsf{Ty}_{\mathcal{S}}\ j\ \Gamma$, $\rho : \mathsf{Sub}_{\mathcal{S}}\ \cdot_{\mathcal{S}}\ \Gamma$.

For example, the syntax S (defined at the end of Section 2) supports a global section functor because syntactic substitutions and terms are in the lowest metatheoretic universe and this universe hierarchy is cumulative. The Set model (defined in Section 3) also supports a global section functor because $\Gamma : \mathsf{Con}_{\mathsf{Set}}\ i$ means $\Gamma : \mathsf{Set}_i$ and $\mathsf{Sub}_{\mathsf{Set}}\ \cdot_{\mathsf{Set}}\ \Gamma = \mathbb{1} \to \Gamma : \mathsf{Set}_i$, and similarly for the second condition.

The *global section functor* $\mathsf{G}^{\mathcal{S}}$ is a pseudomorphism from such an $\mathcal{S}$ to the set model Set of Section 3. It maps a context to the set of closed substitutions into that context. It maps a type to the function sending a closed substitution to the set of closed terms of the type substituted by the input substitution. Substitutions and terms are mapped to postcomposition and substitution by the closed substitution, respectively. We write G instead of $\mathsf{G}^{\mathcal{S}}$ for readability.

$$
\begin{aligned}
\mathsf{G}_{\mathsf{Con}}\,\Gamma &: \underbrace{\mathsf{Con}_{\mathsf{Set}}\,i}_{=\mathsf{Set}_i} && := \mathsf{Sub}_{\mathcal{S}}\ \cdot_{\mathcal{S}}\ \Gamma \\
\mathsf{G}_{\mathsf{Ty}}\,A &: \underbrace{\mathsf{Ty}_{\mathsf{Set}}\,j\,(\mathsf{G}\,\Gamma)}_{=\mathsf{G}\,\Gamma\to\mathsf{Set}_j} && := \lambda\rho\,.\,\mathsf{Tm}_{\mathcal{S}}\ \cdot_{\mathcal{S}}\ (A[\rho]_{\mathcal{S}}) \\
\mathsf{G}_{\mathsf{Sub}}\,\sigma &: \underbrace{\mathsf{Sub}_{\mathsf{Set}}\,(\mathsf{G}\,\Gamma)\,(\mathsf{G}\,\Delta)}_{=\mathsf{G}\,\Gamma\to\mathsf{G}\,\Delta} && := \lambda\rho\,.\,\sigma\circ_{\mathcal{S}}\rho \\
\mathsf{G}_{\mathsf{Tm}}\,t &: \underbrace{\mathsf{Tm}_{\mathsf{Set}}\,(\mathsf{G}\,\Gamma)\,(\mathsf{G}\,A)}_{=(\rho:\mathsf{G}\,\Gamma)\to\mathsf{G}\,A\,\rho} && := \lambda\rho\,.\,t[\rho]_{\mathcal{S}}
\end{aligned}
$$

Note that this pseudomorphism is indeed weak on the empty context: $\mathsf{Sub}_{\mathcal{S}}\ \cdot\ \cdot$ is isomorphic to $\mathbb{1}$ (the empty context in Set) by $\cdot\eta_{\mathcal{S}}$, but not necessarily equal. Similarly, $\mathsf{Sub}_{\mathcal{S}}\ \cdot_{\mathcal{S}}\ (\Gamma \rhd_{\mathcal{S}} A)$ is isomorphic to $(\rho : \mathsf{Sub}_{\mathcal{S}}\ \cdot_{\mathcal{S}}\ \Gamma) \times \mathsf{Tm}_{\mathcal{S}}\ \cdot_{\mathcal{S}}\ (A[\rho]_{\mathcal{S}})$ by comprehension $(\rhd\beta_{1\mathcal{S}}, \rhd\beta_{2\mathcal{S}}, \rhd\eta_{\mathcal{S}})$, but not necessarily equal. In Appendix A we show that G is indeed a pseudomorphism satisfying the conditions in Figure 2.

Remark on comparison maps. In Section 4 we showed that pseudomorphisms support certain comparison maps, for example

$$F_{\Pi.1} : \mathsf{Tm}\,\big(F\,\Gamma \rhd F\,(\Pi\,A\,B)\big)\,\big(\Pi\,(F\,A)\,(F\,B[F_{\rhd.2}])\,[\mathsf{p}]\big)$$

can be defined for any pseudomorphism F. The global section functor gives a good way to show that this comparison map is not an isomorphism in general (as opposed to $F_{\Sigma.1}$). The other direction would be a

$$\mathsf{G}_{\Pi.2} : \underbrace{\mathsf{Tm}_{\mathsf{Set}}\,\big(\mathsf{G}\,\Gamma \rhd_{\mathsf{Set}} \Pi_{\mathsf{Set}}\,(\mathsf{G}\,A)\,(\mathsf{G}\,B[\mathsf{G}_{\rhd.2}])\big)\,\big(\mathsf{G}\,(\Pi_{\mathcal{S}}\,A\,B)[\mathsf{p}]\big)}_{=(\rho:\mathsf{Sub}_{\mathcal{S}}\ \cdot_{\mathcal{S}}\ \Gamma)\times\big((u:\mathsf{Tm}_{\mathcal{S}}\ \cdot_{\mathcal{S}}\ (A[\rho]_{\mathcal{S}}))\to\mathsf{Tm}_{\mathcal{S}}\ \cdot_{\mathcal{S}}\ (B[\rho,u]_{\mathcal{S}})\big)\to\mathsf{Tm}_{\mathcal{S}}\ \cdot_{\mathcal{S}}\ (\Pi_{\mathcal{S}}\ A\ B[\rho]_{\mathcal{S}})},$$

providing a way to turn a metatheoretic function between terms into a term of a function type in $\mathcal{S}$. However if e.g. $S = \mathsf{S}$, $A = \mathsf{Nat}$ and $B = \mathsf{Bool}$, then following Cantor there are more metatheoretic functions from natural numbers to booleans than terms. Similarly, if $\mathsf{G}_{\mathsf{Bool}.2}$ was an isomorphism, it would have an inverse

$$\mathsf{G}_{\mathsf{Bool}.1} : \underbrace{\mathsf{Tm}_{\mathsf{Set}}\,(\mathsf{G}\,\Gamma \triangleright_{\mathsf{Set}} \mathsf{G}\,\mathsf{Bool}_{\mathcal{S}})\,\mathsf{Bool}_{\mathsf{Set}}}_{=(\rho:\mathsf{Sub}_{\mathcal{S}}\,\cdot_{\mathcal{S}}\,\Gamma)\times\mathsf{Tm}_{\mathcal{S}}\,\cdot_{\mathcal{S}}\,\mathsf{Bool}_{\mathcal{S}}\to 2},$$

but if $\mathcal{S}$ is a model without equality reflection where booleans are defined by a quotient then there are more than two terms of type Bool (while internally to $\mathcal{S}$ there are only two elements).

7 Reaping the fruits

Let I be the identity morphism from S to S, which is obviously a strict morphism, hence pseudo.[2] Elimination into gluing along I produces a function whose input is a term t in context Γ and whose output is a term in context Γ extended by $\mathsf{elim}^{\mathsf{P}^{\mathsf{I}}}_{\mathsf{Con}}\,\Gamma$ which expresses that the predicate holds at Γ. The type of the output term says that the predicate holds at A for t. This is the fundamental lemma or parametricity theorem.

$$\mathsf{elim}^{\mathsf{P}^{\mathsf{I}}}_{\mathsf{Tm}} : (t : \mathsf{Tm}_{\mathsf{S}}\,\Gamma\,A) \to \mathsf{Tm}_{\mathsf{S}}\,(\Gamma \triangleright \mathsf{elim}^{\mathsf{P}^{\mathsf{I}}}_{\mathsf{Con}}\,\Gamma)\,\big((\mathsf{elim}^{\mathsf{P}^{\mathsf{I}}}_{\mathsf{Ty}}\,A)[\mathsf{id}, t[\mathsf{p}]]\big)$$

Let us look at the "hello world" example of parametricity, the case where $\Gamma = \cdot$ and $A = \Pi\,(\mathsf{U}\,i)\,(\mathsf{El}\,\mathsf{q} \Rightarrow \mathsf{El}\,\mathsf{q})$. Now using the fact that $\mathsf{elim}^{\mathsf{P}^{\mathsf{I}}}$ is a section, the type of $\mathsf{elim}^{\mathsf{P}^{\mathsf{I}}}_{\mathsf{Tm}}\,t$ computes to

$$\mathsf{Tm}_{\mathsf{S}}\,(\cdot \triangleright \top)\,\Big(\Pi\,(\mathsf{U}\,i)\,\Big(\Pi\,(\mathsf{El}\,\mathsf{q} \Rightarrow \mathsf{U}\,i)\,\Big(\Pi\,(\mathsf{El}\,1)\,\big(\mathsf{El}\,(1\,\$\,0) \Rightarrow \mathsf{El}\,(1\,\$(t\,\$\,2\,\$\,0))\big)\Big)\Big)\Big),$$

where the type is the object theoretic syntax for

$$(A : \mathsf{Set}_i)(C : A \to \mathsf{Set}_i)(a : A) \to C\,a \to C\,(t\,A\,a).$$

Given a fixed type $A : \mathsf{Ty}_{\mathsf{S}}\,i\,\cdot$ and an element $u : \mathsf{Tm}_{\mathsf{S}}\,\cdot\,A$ we have

$$(\mathsf{elim}^{\mathsf{P}^{\mathsf{I}}}_{\mathsf{Tm}}\,t)[\epsilon, \mathsf{tt}]\,\$\,\mathsf{c}\,A\,\$\,\mathsf{lam}\,(\mathsf{c}\,(\mathsf{Id}\,(A[\epsilon])\,0\,u[\epsilon]))\,\$\,u\,\$\,\mathsf{refl}\,u : \mathsf{Tm}\,\cdot\,\big(\mathsf{Id}\,A\,(t\,\$\,c\,A\,\$\,u)\,u\big),$$

that is, we get that for any A and u, $t\,\$\,\mathsf{c}\,A\,\$\,u$ is equal to u.

Eliminating into gluing along $\mathsf{rec}^{\mathsf{Set}}$ (the interpretation into the set model, see end of Section 3) produces Reynolds-style parametricity. It says that if there is an interpretation of the context Γ for which the predicate holds at Γ, the predicate holds at A for the interpretation of t.

$$\mathsf{elim}^{\mathsf{P}^{\mathsf{rec}^{\mathsf{Set}}}}_{\mathsf{Tm}} : (t : \mathsf{Tm}_{\mathsf{S}}\,\Gamma\,A) \to (\gamma : [\![\Gamma]\!]) \times (\bar{\gamma} : \mathsf{elim}^{\mathsf{P}^{\mathsf{rec}^{\mathsf{Set}}}}_{\mathsf{Con}}\,\Gamma\,\gamma) \to \mathsf{elim}^{\mathsf{P}^{\mathsf{rec}^{\mathsf{Set}}}}_{\mathsf{Ty}}\,A\,(\gamma, \bar{\gamma}, [\![t]\!]\,\gamma)$$

Eliminating into gluing along the global section functor G^{S} from the syntax to the set model gives the following.

$$\mathsf{elim}^{\mathsf{P}^{\mathsf{G}}}_{\mathsf{Tm}} : (t : \mathsf{Tm}_{\mathsf{S}}\,\Gamma\,A) \to (\rho : \mathsf{Sub}_{\mathsf{S}}\,\cdot\,\Gamma) \times (\bar{\rho} : \mathsf{elim}^{\mathsf{P}^{\mathsf{G}}}_{\mathsf{Con}}\,\Gamma\,\rho) \to \mathsf{elim}^{\mathsf{P}^{\mathsf{G}}}_{\mathsf{Ty}}\,A\,(\rho, \bar{\rho}, t[\rho])$$

If t is a boolean in the empty context, the type of $\mathsf{elim}^{\mathsf{P}^{\mathsf{G}}}_{\mathsf{Tm}}\,t\,(\mathsf{id}, *)$ is $\mathsf{elim}^{\mathsf{P}^{\mathsf{G}}}_{\mathsf{Ty}}\,\mathsf{Bool}\,(\rho, *, t)$ which is equal to $(b : 2) \times (\mathsf{case}\,b\,\mathsf{true}_{\mathsf{S}}\,\mathsf{false}_{\mathsf{S}} = t)$, i.e. canonicity.

[2] Note that the target of the pseudomorphism needs to have identity types, so technically I is the embedding of the syntax without identity types into the syntax with identity types. Alternatively, we can extend gluing for identity types.

8 Conclusions and further work

In this paper we defined gluing for pseudomorphisms of models of type theory thus generalising parametricity and canonicity. We did not try to derive the most general notion of gluing, e.g. we require that the target model supports $\top$-, Σ-, Id-types in addition to what we have in the domain model. It would have been possible to give a less indexed variant of gluing where $\top$ and Σ are not needed, but Id types (or (F Bool)-indexed inductive families) would be still required to support gluing for Bool. A less indexed variant however would be more tedious to work with because the glued model would involve some metatheoretic equalities.

In the future we would like to generalise our construction to richer type theories having an identity type, inductive and coinductive types. We believe that this is possible without any extra conditions.

Normalisation by evaluation (NBE) for type theory is also defined using a proof-relevant logical predicate [4]. This logical predicate is given by gluing along the Yoneda embedding from the syntax to the presheaf model over the category of contexts and renamings. This is a pseudomorphism, so we obtain a glued model using our method. However, the universe in this model is not what we want. As a second step after gluing, NBE requires the definition of quote and unquote (sometimes called reify and reflect) functions from terms for which the predicate holds to normal forms and from neutral terms to witnesses of the predicate, respectively. We need to include these as part of the universe in the glued model to make the construction work. The predicate for Bool also needs to be adjusted.

We would also like to investigate examples of non-strict pseudomorphisms apart from global section and Yoneda for which the construction in this paper could be useful; for example, to derive canonicity proofs for type theories justified by models other than the set model.

References

1 Andreas Abel, Joakim Öhman, and Andrea Vezzosi. Decidability of conversion for type theory in type theory. *PACMPL*, 2:23:1–23:29, 2017.

2 Thorsten Altenkirch, Martin Hofmann, and Thomas Streicher. Reduction-free normalisation for system F. Unpublished draft, 1997.

3 Thorsten Altenkirch and Ambrus Kaposi. Type theory in type theory using quotient inductive types. In Rastislav Bodik and Rupak Majumdar, editors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 18–29. ACM, 2016. doi:10.1145/2837614.2837638.

4 Thorsten Altenkirch and Ambrus Kaposi. Normalisation by Evaluation for Type Theory, in Type Theory. *Logical Methods in Computer Science*, Volume 13, Issue 4, October 2017. doi:10.23638/LMCS-13(4:1)2017.

5 Michael Artin, Alexander Grothendieck, and Jean-Louis Verdier. *Theorie de Topos et Cohomologie Etale des Schemas I*, volume 269 of *Lecture Notes in Mathematics*. Springer, 1971.

6 Robert Atkey, Neil Ghani, and Patricia Johann. A relationally parametric model of dependent type theory. In Suresh Jagannathan and Peter Sewell, editors, *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 503–516. ACM, 2014. doi:10.1145/2535838.2535852.

7 Jean-Philippe Bernardy, Patrik Jansson, and Ross Paterson. Proofs for Free — Parametricity for Dependent Types. *Journal of Functional Programming*, 22(02):107–152, 2012. doi:10.1017/S0956796812000056.

8. John Cartmell. Generalised algebraic theories and contextual categories. *Annals of Pure and Applied Logic*, 32:209–243, 1986.
9. Pierre Clairambault and Peter Dybjer. The biequivalence of locally cartesian closed categories and Martin-Löf type theories. *Mathematical Structures in Computer Science*, 24(6), 2014.
10. Thierry Coquand. Canonicity and normalisation for Dependent Type Theory. *CoRR*, abs/1810.09367, 2018. `arXiv:1810.09367`.
11. Roy L. Crole. *Categories for types*. Cambridge mathematical textbooks. Cambridge University Press, Cambridge, New York, 1993. URL: `http://opac.inria.fr/record=b1088776`.
12. Peter Dybjer. Internal Type Theory. In *Lecture Notes in Computer Science*, pages 120–134. Springer, 1996.
13. Marcelo Fiore and Alex Simpson. Lambda Definability with Sums via Grothendieck Logical Relations. In Jean-Yves Girard, editor, *Typed Lambda Calculi and Applications*, pages 147–161, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.
14. Marcelo P. Fiore. Semantic analysis of normalisation by evaluation for typed lambda calculus. In *Proceedings of the 4th international ACM SIGPLAN conference on Principles and practice of declarative programming, October 6-8, 2002, Pittsburgh, PA, USA (Affiliated with PLI 2002)*, pages 26–37. ACM, 2002. `doi:10.1145/571157.571161`.
15. Claudio Hermida, Uday S. Reddy, and Edmund P. Robinson. Logical Relations and Parametricity — A Reynolds Programme for Category Theory and Programming Languages. *Electronic Notes in Theoretical Computer Science*, 303(0):149–180, 2014. Proceedings of the Workshop on Algebra, Coalgebra and Topology (WACT 2013). `doi:10.1016/j.entcs.2014.02.008`.
16. Martin Hofmann. Conservativity of Equality Reflection over Intensional Type Theory. In *TYPES 95*, pages 153–164, 1995.
17. B. Jacobs. *Categorical Logic and Type Theory*. Number 141 in Studies in Logic and the Foundations of Mathematics. North Holland, Amsterdam, 1999.
18. Ambrus Kaposi, András Kovács, and Thorsten Altenkirch. Constructing Quotient Inductive-inductive Types. *Proc. ACM Program. Lang.*, 3(POPL):2:1–2:24, January 2019. `doi:10.1145/3290315`.
19. Chung kil Hur and Derek Dreyer. A Kripke logical relation between ML and assembly. *Conference Record of the Annual ACM Symposium on Principles of Programming Languages*, pages 133–146, January 2010. `doi:10.1145/1926385.1926402`.
20. András Kovács. Formalisation of canonicity for type theory in Agda, November 2018. URL: `https://github.com/AndrasKovacs/glue`.
21. J. Lambek and P. J. Scott. *Introduction to higher order categorical logic*. Cambridge University Press, New York, NY, USA, 1986.
22. Gordon D. Plotkin. Lambda-Definability and Logical Relations. Memorandum SAI–RM–4, University of Edinburgh, Edinburgh, Scotland, October 1973.
23. Florian Rabe and Kristina Sojakova. Logical Relations for a Logical Framework. *ACM Trans. Comput. Logic*, 14(4):32:1–32:34, November 2013. `doi:10.1145/2536740.2536741`.
24. John C. Reynolds. Types, Abstraction and Parametric Polymorphism. In R. E. A. Mason, editor, *Information Processing 83, Proceedings of the IFIP 9th World Computer Congress, Paris, September 19-23, 1983*, pages 513–523. Elsevier Science Publishers B. V. (North-Holland), Amsterdam, 1983.
25. Michael Shulman. Univalence for inverse diagrams and homotopy canonicity. *Mathematical Structures in Computer Science*, 25:1203–1277, June 2015. arXiv:1203.3253. `doi:10.1017/S0960129514000565`.
26. Jonathan Sterling and Bas Spitters. Normalization by gluing for free λ-theories. *CoRR*, abs/1809.08646, 2018. `arXiv:1809.08646`.
27. Andrew W. Appel and David Mcallester. An indexed model of recursive types for foundational proof-carrying code. *ACM Trans. Program. Lang. Syst.*, 23:657–683, September 2001. `doi:10.1145/504709.504712`.

A The global section functor is a pseudomorphism

Here we verify the equalities of a pseudomorphism (see Section 4) for the global section functor from a model $\mathcal{S}$ to Set (see Section 3). The definition of the global section functor is given in Section 6, here we repeat it to save turning pages.

$$\mathsf{G}_{\mathsf{Con}}\,\Gamma := \mathsf{Sub}_{\mathcal{S}}\ \cdot_{\mathcal{S}}\ \Gamma$$
$$\mathsf{G}_{\mathsf{Ty}}\,A := \lambda\rho\,.\,\mathsf{Tm}_{\mathcal{S}}\ \cdot_{\mathcal{S}}\ (A[\rho]_{\mathcal{S}})$$
$$\mathsf{G}_{\mathsf{Sub}}\,\sigma := \lambda\rho\,.\,\sigma\circ_{\mathcal{S}}\rho$$
$$\mathsf{G}_{\mathsf{Tm}}\,t := \lambda\rho\,.\,t[\rho]_{\mathcal{S}}$$

$$\mathsf{G}_{\mathsf{id}} : \mathsf{G}\,\mathsf{id}_{\mathcal{S}} = \lambda\rho.\mathsf{id}\circ_{\mathcal{S}}\rho \overset{\mathsf{idl}_{\mathcal{S}}}{=} \lambda\rho.\rho = \mathsf{id}_{\mathsf{Set}}$$
$$\mathsf{G}_{\circ} : \mathsf{G}\,(\sigma\circ_{\mathcal{S}}\delta) = \lambda\rho.(\sigma\circ_{\mathcal{S}}\delta)\circ_{\mathcal{S}}\rho \overset{\mathsf{ass}_{\mathcal{S}}}{=} \lambda\rho.\sigma\circ_{\mathcal{S}}(\delta\circ_{\mathcal{S}}\rho) = \mathsf{G}\,\sigma\circ_{\mathsf{Set}}\mathsf{G}\,\delta$$
$$\mathsf{G}_{[]} : \mathsf{G}\,(A[\delta]_{\mathcal{S}}) = \lambda\rho.\mathsf{Tm}_{\mathcal{S}}\,\cdot\,(A[\delta][\rho]) \overset{[\circ]_{\mathcal{S}}}{=} \lambda\rho.\mathsf{Tm}_{\mathcal{S}}\,\cdot\,(A[\delta\circ\rho]) = (\mathsf{G}\,A)[\mathsf{G}\,\sigma]_{\mathsf{Set}}$$
$$\mathsf{G}_{[]} : \mathsf{G}\,(t[\delta]_{\mathcal{S}}) = \lambda\rho.t[\delta][\rho] \overset{[\circ]_{\mathcal{S}}}{=} \lambda\rho.t[\delta\circ\rho] = (\mathsf{G}\,t)[\mathsf{G}\,\sigma]_{\mathsf{Set}}$$
$$\mathsf{G}_{\cdot} : \mathsf{G}\cdot_{\mathcal{S}}\cong\cdot_{\mathsf{Set}} := (\lambda\rho.*,\lambda_.\epsilon,\mathrm{trivial},\mathrm{trivial})$$
$$\mathsf{G}_{\epsilon} : \mathsf{G}\,\epsilon_{\mathcal{S}} = \lambda\rho.\epsilon\circ\rho \overset{\cdot\eta_{\mathcal{S}}}{=} \lambda\rho.\epsilon = \mathsf{G}_{\cdot.2}\circ_{\mathsf{Set}}\epsilon_{\mathsf{Set}}$$
$$\mathsf{G}_{\triangleright.1} : \mathsf{Sub}_{\mathsf{Set}}\,(\mathsf{G}\,(\Gamma\triangleright_{\mathcal{S}}A))\,(\mathsf{G}\,\Gamma\triangleright_{\mathsf{Set}}\mathsf{G}\,A) := \lambda\rho.(\mathsf{p}\circ\rho,\mathsf{q}[\rho])$$
$$\mathsf{G}_{\triangleright.2} : \mathsf{Sub}_{\mathsf{Set}}\,(\mathsf{G}\,\Gamma\triangleright_{\mathsf{Set}}\mathsf{G}\,A)\,(\mathsf{G}\,(\Gamma\triangleright_{\mathcal{S}}A)) := \lambda(\rho,u).(\rho,_{\mathcal{S}}u)$$
$$\mathsf{G}_{\triangleright.12} : \mathsf{G}_{\triangleright.1}\circ_{\mathsf{Set}}\mathsf{G}_{\triangleright.2} = \lambda(\rho,u).(\mathsf{p}\circ(\rho,_{\mathcal{S}}u),_{\mathcal{S}}\mathsf{q}[\rho,_{\mathcal{S}}u]) \overset{\triangleright\beta_1,\triangleright\beta_2}{=} \lambda(\rho,u).(\rho,u) = \mathsf{id}_{\mathsf{Set}}$$
$$\mathsf{G}_{\triangleright.21} : \mathsf{G}_{\triangleright.2}\circ_{\mathsf{Set}}\mathsf{G}_{\triangleright.1} = \lambda\rho.(\mathsf{p}\circ\rho,_{\mathcal{S}}\mathsf{q}[\rho]) \overset{,\circ_{\mathcal{S}}}{=} \lambda\rho.(\mathsf{p},\mathsf{q})\circ\rho \overset{\triangleright\eta}{=} \lambda\rho.\mathsf{id}\circ\rho \overset{\mathsf{idl}_{\mathcal{S}}}{=} \lambda\rho.\rho = \mathsf{id}_{\mathsf{Set}}$$
$$\mathsf{G}_{\triangleright.1\circ} : \mathsf{G}_{\triangleright.1}\circ_{\mathsf{Set}}\mathsf{G}\,(\sigma^{\uparrow_{\mathcal{S}}}) = \lambda\rho.(\sigma\circ\mathsf{p}\circ\rho,\mathsf{q}[\rho]) \overset{,\circ_{\mathcal{S}}}{=} \lambda\rho.(\sigma\circ\mathsf{p},\mathsf{q})\circ\rho \overset{\mathsf{idl}_{\mathcal{S}},\triangleright\eta_{\mathcal{S}}}{=}$$
$$\lambda\rho.(\sigma\circ\mathsf{p},\mathsf{q})\circ(\mathsf{p}\circ\rho,\mathsf{q}[\rho]) = (\mathsf{G}\,\sigma)^{\uparrow_{\mathsf{Set}}}\circ_{\mathsf{Set}}\mathsf{G}_{\triangleright.1}$$
$$\mathsf{G}_{,} : \mathsf{G}\,(\sigma,_{\mathcal{S}}t) = \lambda\rho.(\sigma,t)\circ\rho \overset{,\circ_{\mathcal{S}}}{=} \rho.(\sigma\circ\rho,t[\rho]) = \mathsf{G}_{\triangleright.2}\circ_{\mathsf{Set}}(\mathsf{G}\,\sigma,_{\mathsf{Set}}\mathsf{G}\,t)$$
$$\mathsf{G}_{\mathsf{p}} : \mathsf{G}\,\mathsf{p}_{\mathcal{S}} = \lambda\rho.\mathsf{p}\circ\rho = \lambda\rho.(\mathsf{p}\circ\rho,\mathsf{q}[\rho]).1 = \mathsf{p}_{\mathsf{Set}}\circ_{\mathsf{Set}}\mathsf{G}_{\triangleright.1}$$
$$\mathsf{G}_{\mathsf{q}} : \mathsf{G}\,\mathsf{q}_{\mathcal{S}} = \lambda\rho.\mathsf{q}[\rho] = \lambda\rho.(\mathsf{p}\circ\rho,\mathsf{q}[\rho]).2 = \mathsf{q}_{\mathsf{Set}}[\mathsf{G}_{\triangleright.1}]_{\mathsf{Set}}$$

B Full version of if_{P} and equalities in gluing

if_{P} is part of the glued displayed model P, see Section 5, Figure 3. Its definition is the following including the omitted _ arguments.

$$\begin{aligned}
&\mathsf{if}_{\mathsf{P}}\,C_{\mathsf{P}}\,t_{\mathsf{P}}\,u_{\mathsf{P}}\,v_{\mathsf{P}} := \\
&\quad \mathsf{J}\,\Big(C_{\mathsf{P}}\big[F_{\triangleright.2}\circ(\mathsf{p}^3,1),(2,(\mathsf{projl}\,t_{\mathsf{P}}[\mathsf{p}^2],0)),F\,\big(\mathsf{if}_{\mathcal{S}}\,(C[\mathsf{p}^2,\mathsf{q}])\,\mathsf{q}\,(u[\mathsf{p}])\,(v[\mathsf{p}])\big)[F_{\triangleright.2}\circ(\mathsf{p}^3,1)]\big]\Big) \\
&\qquad \Big(\mathsf{if}\,\big(C_{\mathsf{P}}\big[F_{\triangleright.2}\circ(\mathsf{p}^2,w),(1,(0,\mathsf{refl}\,w)),F\,\big(\mathsf{if}_{\mathcal{S}}\,(C[\mathsf{p}^2,\mathsf{q}])\,\mathsf{q}\,(u[\mathsf{p}])\,(v[\mathsf{p}])\big)[F_{\triangleright.2}\circ(\mathsf{p}^2,w)]\big]\big) \\
&\qquad\quad (\mathsf{projl}\,t_{\mathsf{P}})\,u_{\mathsf{P}}\,v_{\mathsf{P}}\,\Big) \\
&\qquad (\mathsf{projr}\,t_{\mathsf{P}})
\end{aligned}$$

where w abbreviates $\mathsf{if}\,(F\,\mathsf{Bool}_{\mathcal{S}}[\mathsf{p}^2])\,0\,(F\,\mathsf{true}_{\mathcal{S}}[\mathsf{p}^2])\,(F\,\mathsf{false}_{\mathcal{S}}[\mathsf{p}^2])$.

Here we check that the P satisfies all the equalities of displayed models. We note that $\sigma_{\mathsf{P}}{}^{\uparrow_{\mathsf{P}}} = \big(\sigma_{\mathsf{P}}[\mathsf{p}\circ F_{\triangleright.1}\circ\mathsf{p},\mathsf{projl}\,\mathsf{q}],\mathsf{projr}\,\mathsf{q}\big)$.

$$
\begin{aligned}
&\mathsf{idl_P} &&: \mathsf{id_P} \circ_\mathsf{P} \sigma_\mathsf{P} = 0[F\,\sigma \circ \mathsf{p}, \sigma_\mathsf{P}] = \sigma_\mathsf{P}\\
&\mathsf{idr_P} &&: \sigma_\mathsf{P} \circ_\mathsf{P} \mathsf{id_P} = \sigma_\mathsf{P}[F\,\mathsf{id} \circ \mathsf{p}, 0] = \sigma_\mathsf{P}[\mathsf{p},\mathsf{q}] = \sigma_\mathsf{P}[\mathsf{id}] = \sigma_\mathsf{P}\\
&\mathsf{ass_P} &&: (\sigma_\mathsf{P} \circ_\mathsf{P} \delta_\mathsf{P}) \circ_\mathsf{P} \nu_\mathsf{P} = \sigma_\mathsf{P}[F\,\delta \circ \mathsf{p}, \delta_\mathsf{P}][F\,\nu \circ \mathsf{p}, \nu_\mathsf{P}] =\\
& && \quad \sigma_\mathsf{P}[F\,(\delta \circ_\mathcal{S} \nu) \circ \mathsf{p}, \delta_\mathsf{P}[F\,\nu \circ \mathsf{p}, \nu_\mathsf{P}]] = \sigma_\mathsf{P} \circ_\mathsf{P} (\delta_\mathsf{P} \circ_\mathsf{P} \nu_\mathsf{P})\\
&[\mathsf{id}]_\mathsf{P} &&: A_\mathsf{P}[\mathsf{id_P}]_\mathsf{P} = A_\mathsf{P}[F\,\mathsf{id} \circ \mathsf{p}^2, \mathsf{q}[\mathsf{p}], \mathsf{q}] = A_\mathsf{P}[(\mathsf{p},\mathsf{q}) \circ \mathsf{p}, \mathsf{q}] = A_\mathsf{P}[\mathsf{id} \circ\ \mathsf{p}, \mathsf{q}] = A_\mathsf{P}[\mathsf{id}] = A_\mathsf{P}\\
&[\circ]_\mathsf{P} &&: A_\mathsf{P}[\sigma_\mathsf{P} \circ_\mathsf{P} \delta_\mathsf{P}]_\mathsf{P} = A_\mathsf{P}[F\,(\sigma \circ_\mathcal{S} \delta) \circ \mathsf{p}^2, \sigma_\mathsf{P}[F\,\delta \circ \mathsf{p}, \delta_\mathsf{P}][\mathsf{p}], \mathsf{q}] =\\
& && \quad A_\mathsf{P}[F\,\sigma \circ \mathsf{p}^2, \sigma_\mathsf{P}[\mathsf{p}], \mathsf{q}][F\,\delta \circ \mathsf{p}^2, \delta_\mathsf{P}[\mathsf{p}], \mathsf{q}] = A_\mathsf{P}[\sigma_\mathsf{P}]_\mathsf{P}[\delta_\mathsf{P}]_\mathsf{P}\\
&[\mathsf{id}]_\mathsf{P} &&: t_\mathsf{P}[\mathsf{id_P}]_\mathsf{P} = t_\mathsf{P}[F\,\mathsf{id} \circ \mathsf{p}, \mathsf{q}] = t_\mathsf{P}[\mathsf{p},\mathsf{q}] = t_\mathsf{P}[\mathsf{id}] = t_\mathsf{P}\\
&[\circ]_\mathsf{P} &&: t_\mathsf{P}[\sigma_\mathsf{P} \circ_\mathsf{P} \delta_\mathsf{P}]_\mathsf{P} = t_\mathsf{P}[F\,(\sigma \circ \delta) \circ \mathsf{p}, \sigma_\mathsf{P}[F\,\delta \circ \mathsf{p}, \delta_\mathsf{P}]] =\\
& && \quad t_\mathsf{P}[F\,\sigma \circ \mathsf{p}, \sigma_\mathsf{P}][F\,\delta \circ \mathsf{p}, \delta_\mathsf{P}] = t_\mathsf{P}[\sigma_\mathsf{P}]_\mathsf{P}[\delta_\mathsf{P}]_\mathsf{P}\\
&\epsilon\eta_\mathsf{P} &&: (\delta_\mathsf{P} : \mathsf{Sub_P}\,\Gamma_\mathsf{P}\,\cdot_\mathsf{P}) = (\delta_\mathsf{P} : \mathsf{Tm}\,(F\,\Gamma \rhd \Gamma_\mathsf{P})\,\top) \overset{\cdot\eta}{=} \mathsf{tt} = \epsilon_\mathsf{P}\\
&\rhd\beta_{1\mathsf{P}} &&: \mathsf{p_P} \circ_\mathsf{P} (\sigma_\mathsf{P},_\mathsf{P} t_\mathsf{P}) = (\mathsf{projl}\,\mathsf{q})[F\,(\sigma,_\mathcal{S} t) \circ \mathsf{p}, (\sigma_\mathsf{P}, t_\mathsf{P})] =\\
& && \quad \mathsf{projl}\,(\mathsf{q}[F\,(\sigma,_\mathcal{S} t) \circ \mathsf{p}, (\sigma_\mathsf{P}, t_\mathsf{P})]) = \mathsf{projl}\,(\sigma_\mathsf{P}, t_\mathsf{P}) = \sigma_\mathsf{P}\\
&\rhd\beta_{2\mathsf{P}} &&: \mathsf{q_P}[\sigma_\mathsf{P},_\mathsf{P} t_\mathsf{P}]_\mathsf{P} = (\mathsf{projr}\,\mathsf{q})[F\,(\sigma,_\mathcal{S} t) \circ \mathsf{p}, (\sigma_\mathsf{P}, t_\mathsf{P})] =\\
& && \quad \mathsf{projr}\,(\mathsf{q}[F\,(\sigma,_\mathcal{S} t) \circ \mathsf{p}, (\sigma_\mathsf{P}, t_\mathsf{P})]) = \mathsf{projr}\,(\sigma_\mathsf{P}, t_\mathsf{P}) = t_\mathsf{P}\\
&\rhd\eta_\mathsf{P} &&: (\mathsf{p_P},_\mathsf{P} \mathsf{q_P}) = (\mathsf{projl}\,\mathsf{q}, \mathsf{projr}\,\mathsf{q}) \overset{\Sigma\eta}{=} \mathsf{q} = \mathsf{id_P}\\
&,\circ_\mathsf{P} &&: (\sigma_\mathsf{P},_\mathsf{P} t_\mathsf{P}) \circ_\mathsf{P} \delta_\mathsf{P} = (\sigma_\mathsf{P}, t_\mathsf{P})[F\,\delta \circ \mathsf{p}, \delta_\mathsf{P}] \overset{,[]}{=} (\sigma_\mathsf{P}[F\,\delta \circ \mathsf{p}, \delta_\mathsf{P}], t_\mathsf{P}[F\,\delta \circ \mathsf{p}, \delta_\mathsf{P}]) =\\
& && \quad (\sigma_\mathsf{P} \circ_\mathsf{P} \delta_\mathsf{P},_\mathsf{P} t_\mathsf{P}[\delta_\mathsf{P}]_\mathsf{P})\\
&\Pi\beta_\mathsf{P} &&: \mathsf{app_P}\,(\mathsf{lam_P}\,t_\mathsf{P}) =\\
& && \quad \big(\mathsf{app}\,(\mathsf{app}\,(\mathsf{lam}\,(\mathsf{lam}\,(t_\mathsf{P}[F_{\rhd.2} \circ (\mathsf{p}^3, 1), (2,0)])))))\big)\\
& && \qquad \big[\mathsf{p} \circ F_{\rhd.1} \circ \mathsf{p}, \mathsf{projl}\,0, 0[F_{\rhd.1} \circ \mathsf{p}], \mathsf{projr}\,0\big] \overset{\Pi\beta}{=}\\
& && \quad t_\mathsf{P}[F_{\rhd.2} \circ (\mathsf{p}^3, 1), (2,0)]\big[\mathsf{p} \circ F_{\rhd.1} \circ \mathsf{p}, \mathsf{projl}\,0, 0[F_{\rhd.1} \circ \mathsf{p}], \mathsf{projr}\,0\big] =\\
& && \quad t_\mathsf{P}[\mathsf{p}, (\mathsf{projl}\,0, \mathsf{projr}\,0)] = t_\mathsf{P}[\mathsf{id}] = t_\mathsf{P}\\
&\Pi\eta_\mathsf{P} &&: \mathsf{lam_P}\,(\mathsf{app_P}\,t_\mathsf{P}) =\\
& && \quad \mathsf{lam}\,\Big(\mathsf{lam}\,\big(\big(\mathsf{app}\,(\mathsf{app}\,t_\mathsf{P})\big)\big[\mathsf{p} \circ F_{\rhd.1} \circ \mathsf{p}, \mathsf{projl}\,0, 0[F_{\rhd.1} \circ \mathsf{p}], \mathsf{projr}\,0\big]\\
& && \qquad\qquad \big[F_{\rhd.2} \circ (\mathsf{p}^3, 1), (2,0)\big]\big)\Big) =\\
& && \quad \mathsf{lam}\,\big(\mathsf{lam}\,\big((\mathsf{app}\,(\mathsf{app}\,t_\mathsf{P}))[\mathsf{p}^3, 2, 1, 0]\big)\big) = \mathsf{lam}\,(\mathsf{lam}\,((\mathsf{app}\,(\mathsf{app}\,t_\mathsf{P}))[\mathsf{id}])) \overset{\Pi\eta_\mathcal{S}}{=} t_\mathsf{P}\\
&\Pi[]_\mathsf{P} &&: (\Pi_\mathsf{P}\,A_\mathsf{P}\,B_\mathsf{P})[\sigma_\mathsf{P}]_\mathsf{P} =\\
& && \quad \Pi\,\big(F\,A[F\,\sigma \circ \mathsf{p}^2]\big)\\
& && \qquad \Big(\Pi\,\big(A_\mathsf{P}\big[F\,\sigma \circ \mathsf{p}^2, \sigma_\mathsf{P}[\mathsf{p}], \mathsf{q}\big][\mathsf{p}^2, \mathsf{q}]\big)\\
& && \qquad\quad \big(B_\mathsf{P}\big[F_{\rhd.2} \circ (F\,\sigma \circ \mathsf{p}^4, 1), (\sigma_\mathsf{P}[\mathsf{p}^3], 0),\\
& && \qquad\qquad F\,(\mathsf{app}\,\mathsf{q})[F_{\rhd.2} \circ (F_{\rhd.2} \circ (F\,\sigma \circ \mathsf{p}^4, 2), 1)]\big]\big)\Big) =\\
& && \quad \Pi\,\big(F\,(A[\sigma])[\mathsf{p}^2]\big)\\
& && \qquad \Big(\Pi\,\big(A_\mathsf{P}[\sigma_\mathsf{P}]_\mathsf{P}[\mathsf{p}^2, \mathsf{q}]\big)\\
& && \qquad\quad \big(B_\mathsf{P}\big[F_{\rhd.2} \circ (F\,\sigma)^\uparrow \circ F_{\rhd.1} \circ \mathsf{p}^2, \big(\sigma_\mathsf{P}[\mathsf{p} \circ F_{\rhd.1} \circ \mathsf{p}^2, \mathsf{projl}\,1], \mathsf{projr}\,1\big), \mathsf{q}\big]\\
& && \qquad\qquad \big[F_{\rhd.2} \circ (\mathsf{p}^4, 1), (3,0), F\,(\mathsf{app}\,\mathsf{q})[F_{\rhd.2} \circ (F_{\rhd.2} \circ (\mathsf{p}^4, 2), 1)]\big]\big)\Big) =\\
& && \quad \Pi_\mathsf{P}\,(A_\mathsf{P}[\sigma_\mathsf{P}]_\mathsf{P})\,(B_\mathsf{P}[\sigma_\mathsf{P}{}^{\uparrow_\mathsf{P}}]_\mathsf{P})
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{lam}[]_{\mathsf{P}} &&:(\mathsf{lam}_{\mathsf{P}}\, t_{\mathsf{P}})[\sigma_{\mathsf{P}}]_{\mathsf{P}} = \mathsf{lam}\,(\mathsf{lam}\,(t_{\mathsf{P}}[F_{\triangleright.2}\circ(F\,\sigma\circ\mathsf{p}^3,1),(\sigma_{\mathsf{P}}[\mathsf{p}^2],0)])) = \\
&&&\quad \mathsf{lam}_{\mathsf{P}}\,(t_{\mathsf{P}}[\sigma_{\mathsf{P}}{}^{\uparrow_{\mathsf{P}}}]_{\mathsf{P}}) \\
&\Sigma\beta_{1\mathsf{P}} &&:\mathsf{projl}_{\mathsf{P}}\,(u_{\mathsf{P}},_{\mathsf{P}}\, v_{\mathsf{P}}) = \mathsf{projl}\,(u_{\mathsf{P}},v_{\mathsf{P}}) = u_{\mathsf{P}} \\
&\Sigma\beta_{2\mathsf{P}} &&:\mathsf{projr}_{\mathsf{P}}\,(u_{\mathsf{P}},_{\mathsf{P}}\, v_{\mathsf{P}}) = \mathsf{projr}\,(u_{\mathsf{P}},v_{\mathsf{P}}) = v_{\mathsf{P}} \\
&\Sigma\eta_{\mathsf{P}} &&:(\mathsf{projl}_{\mathsf{P}}\, t_{\mathsf{P}},_{\mathsf{P}}\,\mathsf{projr}_{\mathsf{P}}\, t_{\mathsf{P}}) = (\mathsf{projl}\, t_{\mathsf{P}},\mathsf{projr}\, t_{\mathsf{P}}) = t_{\mathsf{P}} \\
&\Sigma[]_{\mathsf{P}} &&:(\Sigma_{\mathsf{P}}\, A_{\mathsf{P}}\, B_{\mathsf{P}})[\sigma_{\mathsf{P}}]_{\mathsf{P}} = \\
&&&\quad \Sigma\,\big(A_{\mathsf{P}}\big[F\,\sigma\circ\mathsf{p}^2,\sigma_{\mathsf{P}}[\mathsf{p}],F\,(\mathsf{projl}\,\mathsf{q})[F_{\triangleright.2}\circ(F\,\sigma\circ\mathsf{p}^2,\mathsf{q})]\big]\big) \\
&&&\quad\quad \big(B_{\mathsf{P}}\big[F_{\triangleright.2}\circ\big(F\,\sigma\circ\mathsf{p}^3,F\,(\mathsf{projl}\,\mathsf{q})[F_{\triangleright.2}\circ(F\,\sigma\circ\mathsf{p}^3,1)]\big),(\sigma_{\mathsf{P}}[\mathsf{p}^2],0), \\
&&&\quad\quad\quad F\,(\mathsf{projr}\,\mathsf{q})[F_{\triangleright.2}\circ(F\,\sigma\circ\mathsf{p}^3,1)]\big]\big) = \\
&&&\quad \Sigma\,\big(A_{\mathsf{P}}\big[F\,\sigma\circ\mathsf{p}^2,\sigma_{\mathsf{P}}[\mathsf{p}],F\,(\mathsf{projl}\,\mathsf{q})[F_{\triangleright.2}\circ(\mathsf{p}^2,\mathsf{q})]\big]\big) \\
&&&\quad\quad \big(B_{\mathsf{P}}\big[F_{\triangleright.2}\circ\big(F\,\sigma\circ\mathsf{p}^3,F\,(\mathsf{projl}\,\mathsf{q})[F_{\triangleright.2}\circ(\mathsf{p}^3,1)]\big),(\sigma_{\mathsf{P}}[\mathsf{p}^2],0), \\
&&&\quad\quad\quad F\,(\mathsf{projr}\,\mathsf{q})[F_{\triangleright.2}\circ(\mathsf{p}^3,1)]\big]\big) = \\
&&&\quad \Sigma_{\mathsf{P}}\,(A_{\mathsf{P}}[\sigma_{\mathsf{P}}]_{\mathsf{P}})\,(B_{\mathsf{P}}[\sigma_{\mathsf{P}}{}^{\uparrow_{\mathsf{P}}}]_{\mathsf{P}}) \\
&,[]_{\mathsf{P}} &&:(u_{\mathsf{P}},_{\mathsf{P}}\, v_{\mathsf{P}})[\sigma_{\mathsf{P}}]_{\mathsf{P}} = (u_{\mathsf{P}}[F\,\sigma\circ\mathsf{p},\sigma_{\mathsf{P}}],v_{\mathsf{P}}[F\,\sigma\circ\mathsf{p},\sigma_{\mathsf{P}}]) = (u_{\mathsf{P}}[\sigma_{\mathsf{P}}]_{\mathsf{P}},_{\mathsf{P}}\, v_{\mathsf{P}}[\sigma_{\mathsf{P}}]_{\mathsf{P}}) \\
&\top\eta_{\mathsf{P}} &&:(t_{\mathsf{P}}:\mathsf{Tm}_{\mathsf{P}}\,\Gamma_{\mathsf{P}}\,\top_{\mathsf{P}}) = (t_{\mathsf{P}}:\mathsf{Tm}\,(F\,\Gamma\triangleright\Gamma_{\mathsf{P}})\,\top) \overset{\top\eta}{=} \mathsf{tt} = \mathsf{tt}_{\mathsf{P}} \\
&\top[]_{\mathsf{P}} &&:\top_{\mathsf{P}}[\sigma_{\mathsf{P}}]_{\mathsf{P}} = \top = \top_{\mathsf{P}} \\
&\mathsf{tt}[]_{\mathsf{P}} &&:\mathsf{tt}_{\mathsf{P}}[\sigma_{\mathsf{P}}]_{\mathsf{P}} = \mathsf{tt} = \mathsf{tt}_{\mathsf{P}} \\
&\mathsf{U}\beta_{\mathsf{P}} &&:\mathsf{El}_{\mathsf{P}}\,(\mathsf{c}_{\mathsf{P}}\, A_{\mathsf{P}}) = \mathsf{El}\,(\mathsf{app}\,(\mathsf{lam}\,(\mathsf{c}\, A_{\mathsf{P}}))) \overset{\Pi\beta}{=} \mathsf{El}\,(\mathsf{c}\, A_{\mathsf{P}}) \overset{\mathsf{U}\beta}{=} A_{\mathsf{P}} \\
&\mathsf{U}\eta_{\mathsf{P}} &&:\mathsf{c}_{\mathsf{P}}\,(\mathsf{El}_{\mathsf{P}}\, a_{\mathsf{P}}) = \mathsf{lam}\,(\mathsf{c}\,(\mathsf{El}\,(\mathsf{app}\, a_{\mathsf{P}}))) \overset{\mathsf{El}\eta}{=} \mathsf{lam}\,(\mathsf{app}\, a_{\mathsf{P}}) \overset{\Pi\eta}{=} a_{\mathsf{P}} \\
&\mathsf{U}[]_{\mathsf{P}} &&:(\mathsf{U}_{\mathsf{P}}\, i)[\sigma_{\mathsf{P}}]_{\mathsf{P}} = F\,(\mathsf{El}_{\mathcal{S}}\,\mathsf{q})[F_{\triangleright.2}\circ(F\,\sigma\circ\mathsf{p}^2,\mathsf{q})]\Rightarrow\mathsf{U}\, i = \\
&&&\quad F\,(\mathsf{El}_{\mathcal{S}}\,\mathsf{q})[F_{\triangleright.2}\circ(F\,\sigma)^{\uparrow}\circ(\mathsf{p}^2,\mathsf{q})]\Rightarrow\mathsf{U}\, i = \\
&&&\quad F\,(\mathsf{El}_{\mathcal{S}}\,\mathsf{q})[F\,(\sigma^{\uparrow})\circ F_{\triangleright.2}\circ(\mathsf{p}^2,\mathsf{q})]\Rightarrow\mathsf{U}\, i = F\,(\mathsf{El}_{\mathcal{S}}\,\mathsf{q})[F_{\triangleright.2}\circ(\mathsf{p}^2,\mathsf{q})]\Rightarrow\mathsf{U}\, i = \mathsf{U}_{\mathsf{P}}\, i \\
&\mathsf{El}[]_{\mathsf{P}} &&:(\mathsf{El}_{\mathsf{P}}\, a_{\mathsf{P}})[\sigma_{\mathsf{P}}]_{\mathsf{P}} = (\mathsf{El}\,(\mathsf{app}\, a_{\mathsf{P}}))[F\,\sigma\circ\mathsf{p}^2,\sigma_{\mathsf{P}}[\mathsf{p}],\mathsf{q}] \overset{\mathsf{El}[]}{=} \\
&&&\quad \mathsf{El}\,((\mathsf{app}\, a_{\mathsf{P}})[F\,\sigma\circ\mathsf{p}^2,\sigma[\mathsf{p}],\mathsf{q}]) \overset{\mathsf{app}[]}{=} \mathsf{El}\,(\mathsf{app}\,(a_{\mathsf{P}}[F\,\sigma\circ\mathsf{p},\sigma_{\mathsf{P}}])) = \mathsf{El}_{\mathsf{P}}\,(a_{\mathsf{P}}[\sigma_{\mathsf{P}}]_{\mathsf{P}}) \\
&\mathsf{Bool}[]_{\mathsf{P}} &&:\mathsf{Bool}_{\mathsf{P}}[\sigma_{\mathsf{P}}]_{\mathsf{P}} = \\
&&&\quad \Sigma\,\mathsf{Bool}\,\big(\mathsf{Id}\,(F\,\mathsf{Bool}[F\,\sigma\circ\mathsf{p}^3]) \\
&&&\quad\quad\quad \big(\mathsf{if}\,(F\,\mathsf{Bool}[F\,\sigma\circ\mathsf{p}^4])\,0\,(F\,\mathsf{true}[F\,\sigma\circ\mathsf{p}^3])\,(F\,\mathsf{false}[F\,\sigma\circ\mathsf{p}^3])\big)\,1\big) = \\
&&&\quad \Sigma\,\mathsf{Bool}\,\big(\mathsf{Id}\,(F\,\mathsf{Bool}[\mathsf{p}^3])\,\big(\mathsf{if}\,(F\,\mathsf{Bool}[\mathsf{p}^4])\,0\,(F\,\mathsf{true}[\mathsf{p}^3])\,(F\,\mathsf{false}[\mathsf{p}^3])\big)\,1\big) = \mathsf{Bool}_{\mathsf{P}} \\
&\mathsf{true}[]_{\mathsf{P}} &&:\mathsf{true}_{\mathsf{P}}[\sigma_{\mathsf{P}}]_{\mathsf{P}} = \big(\mathsf{true},\mathsf{refl}\,(F\,(\mathsf{true}_{\mathcal{S}}[\sigma])[\mathsf{p}])\big) \overset{\mathsf{true}[]_{\mathcal{S}}}{=} (\mathsf{true},\mathsf{refl}\,(F\,\mathsf{true}_{\mathcal{S}}[\mathsf{p}])) = \mathsf{true}_{\mathsf{P}} \\
&\mathsf{false}[]_{\mathsf{P}} &&:\mathsf{false}_{\mathsf{P}}[\sigma_{\mathsf{P}}]_{\mathsf{P}} = \big(\mathsf{false},\mathsf{refl}\,(F\,(\mathsf{false}_{\mathcal{S}}[\sigma])[\mathsf{p}])\big) \overset{\mathsf{false}[]_{\mathcal{S}}}{=} (\mathsf{false},\mathsf{refl}\,(F\,\mathsf{false}_{\mathcal{S}}[\mathsf{p}])) = \mathsf{false}_{\mathsf{P}} \\
&\mathsf{if}[]_{\mathsf{P}} &&:(\mathsf{if}_{\mathsf{P}}\,_\, t_{\mathsf{P}}\, u_{\mathsf{P}}\, v_{\mathsf{P}})[\sigma_{\mathsf{P}}]_{\mathsf{P}} = \\
&&&\quad \big(\mathsf{J}\,_\,(\mathsf{if}\,_\,(\mathsf{projl}\, t_{\mathsf{P}})\, u_{\mathsf{P}}\, v_{\mathsf{P}})\,(\mathsf{projr}\, t_{\mathsf{P}})\big)[F\,\sigma\circ\mathsf{p},\sigma_{\mathsf{P}}] \overset{\mathsf{J}[],\mathsf{if}[],\mathsf{projl}[],\mathsf{projr}[]}{=} \\
&&&\quad \big(\mathsf{J}\,_\,(\mathsf{if}\,_\,(\mathsf{projl}\,(t_{\mathsf{P}}[\sigma_{\mathsf{P}}]_{\mathsf{P}}))\,(u_{\mathsf{P}}[\sigma_{\mathsf{P}}]_{\mathsf{P}})\,(v_{\mathsf{P}}[\sigma_{\mathsf{P}}]_{\mathsf{P}}))\,(\mathsf{projr}\,(t_{\mathsf{P}}[\sigma_{\mathsf{P}}]_{\mathsf{P}}))\big) = \\
&&&\quad \mathsf{if}_{\mathsf{P}}\,_\,(t_{\mathsf{P}}[\sigma_{\mathsf{P}}]_{\mathsf{P}})\,(u_{\mathsf{P}}[\sigma_{\mathsf{P}}]_{\mathsf{P}})\,(v_{\mathsf{P}}[\sigma_{\mathsf{P}}]_{\mathsf{P}})
\end{aligned}
$$

The Discriminating Power of the Let-In Operator in the Lazy Call-by-Name Probabilistic λ-Calculus

Simona Kašterović

Faculty of Technical Sciences, University of Novi Sad
Trg Dositeja Obradovića 6, 21000 Novi Sad, Serbia
http://imft.ftn.uns.ac.rs/~simona/
simona.k@uns.ac.rs

Michele Pagani

IRIF, University Paris Diderot - Paris 7, France
https://www.irif.fr/~michele/
pagani@irif.fr

Abstract

We consider the notion of probabilistic applicative bisimilarity (PAB), recently introduced as a behavioural equivalence over a probabilistic extension of the untyped λ-calculus. Alberti, Dal Lago and Sangiorgi have shown that PAB is not fully abstract with respect to the context equivalence induced by the lazy call-by-name evaluation strategy. We prove that extending this calculus with a let-in operator allows for achieving the full abstraction. In particular, we recall Larsen and Skou's testing language, which is known to correspond with PAB, and we prove that every test is representable by a context of our calculus.

2012 ACM Subject Classification Theory of computation → Lambda calculus

Keywords and phrases probabilistic lambda calculus, bisimulation, Howe's technique, context equivalence, testing

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.26

Acknowledgements We wish to thank the anonymous reviewers for their valuable suggestions, helping us to improve the paper.

1 Introduction

We consider the probabilistic extension $\Lambda_\oplus$ of the untyped λ-calculus, obtained by adding a probabilistic choice primitive $M \oplus N$ representing a term evaluating to M or N with equal probability. This calculus provides a useful although quite simple framework for importing tools and results from the standard theory of the λ-calculus to probabilistic programming.

As well-known, the choice of an evaluation strategy for $\Lambda_\oplus$ plays a crucial role, even for strongly normalising terms. Consider a function $\lambda x.F$ applied to a probabilistic term $M \oplus N$: if we adopt a call-by-name policy, cbn by short, the whole term $M \oplus N$ would be passed to the calling parameter x *before* actually performing the choice between M and N, while in a call-by-value strategy, cbv by short, we first chose between M and N and *then* pass the value associated with this choice to x. If the evaluation of F calls n times the parameter x, then the cbn strategy performs n independent choices between M and N, while the cbv strategy copies n times the result of one single choice. In linear logic semantics [11], this phenomenon can be described by precising that the application is a bilinear function in cbv (so $(\lambda x.F)(M \oplus N)$ is equivalent to $((\lambda x.F)M) \oplus ((\lambda x.F)N)$), while it is not linear in the argument position in cbn (see discussion at Example 3).

In probabilistic programming it is worthwhile to have a cbv operator even in a cbn language, as the most of the randomised algorithms need to sample from a distribution and passing to a sub-procedure the *value* of this sample rather than the *whole distribution*.

© Simona Kašterović and Michele Pagani;
licensed under Creative Commons License CC-BY

4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019).
Editor: Herman Geuvers; Article No. 26; pp. 26:1–26:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Consider for example the randomised quicksort: this algorithm takes a pivot randomly from an array and it passes it to the partitioning procedure, which uses this pivot several times. The algorithm would be unsound if we allow to make different choices each time the partitioning procedure calls for the same pivot. In [10] the authors enrich the cbn probabilistic PCF with a let-in operator, restricted to the ground values, so that let $x = M \oplus N$ in F behaves like a cbv application of $\lambda x.F$ to $M \oplus N$. In a continuous framework this kind of operator is usually called *sampling* (e.g. [15]), but this is just a different terminology for the same computation mechanism: sampling a value from a distribution before passing it to a parameter.

Both calling policies (cbn and cbv) can be declined with a further attribute which is Abramsky's lazyness [1]: a reduction strategy is *lazy* (sometimes called also *weak*) whenever it does not evaluate the body of a function, i.e. it does not reduce a β-redex under the scope of a λ-abstraction. This notion has been presented in order to provide a formal model of the evaluation mechanism of the lazy functional programming languages.

Two probabilistic programs are context equivalent if they have the same probability of converging to a value in all contexts. Of course, this notion depends on which reduction strategy has been chosen. The prototypical example of diverging term $\mathbf{\Omega} \stackrel{\text{def}}{=} (\lambda x.xx)(\lambda x.xx)$ is context equivalent with $\lambda x.\mathbf{\Omega}$ for a non-lazy strategy, while the two terms can be trivially distinguished by a lazy strategy as $\lambda x.\mathbf{\Omega}$ is a value for such a reduction. Similarly, the term $(\lambda xy.y)\mathbf{\Omega}$ is equivalent to $\mathbf{\Omega}$ for cbv, but it is converging for the cbn policy (lazy or not), because the reduction step $(\lambda xy.y)\mathbf{\Omega} \to \lambda y.y$ is admitted.

One of the major contribution of the already mentioned [1] has been to use the notion of bisimilarity in order to study the context equivalence of the lazy cbn λ-calculus. The idea is to consider a reduction strategy as a labelled transition system where the states and labels of the system are the λ-terms and a transition labelled by a term P goes from a term M to a value M' whenever M' is the result of evaluating the application MP. The benefit of this setting is to be able to transport into λ-calculus the whole theory of bisimilarity (called in [1] *applicative bisimilarity*) and its associated coinduction reasoning, which is one of the main tools for comparing processes in concurrency theory. Basically, two terms M and N are applicative bisimilar whenever their applications MP and NP are applicative bisimilar for any argument P. Abramsky proved that applicative bisimilarity is sound with respect to lazy cbn context equivalence (i.e. the former implies the latter), but it is not fully abstract (there are context equivalent terms that are not bisimilar).

Abramsky's applicative bisimilarity has been recently lifted to $\Lambda_\oplus$ by Dal Lago and his co-authors [3, 7]. The transition system becomes now a Markov Chain (here Definition 14) on the top of it one can define a notion of probabilistic applicative bisimilarity (PAB). The paper [7] considers a lazy cbn reduction strategy, while [3] focuses on the (lazy) cbv strategy. In both settings, PAB is proven sound with respect to the associated context equivalence, but, surprisingly, the cbv bisimilarity is also fully abstract, while the lazy cbn is not. Our paper shows that adding the let-in operator mentioned before is enough for recovering the full abstraction even for the lazy cbn.

Let us discuss more in detail the problem with the lazy cbn operation semantics. The two terms $\lambda xy.(x \oplus y)$ and $(\lambda xy.x) \oplus (\lambda xy.y)$ are context equivalent but not bisimilar (Example 7). The difference is between a process giving a value *allowing* two choices and a process giving two values *after* a choice (see Figure 4 to have a pictorial representation of the two processes). The cbn contexts are not able to discriminate such a subtle difference while bisimilarity does (Examples 16 and 23). In [3] the authors show a cbv context discriminating a variant of these two terms and they conjecture that a kind of sequencing operator can recover the full abstraction for the lazy cbn : our paper proves this conjecture.

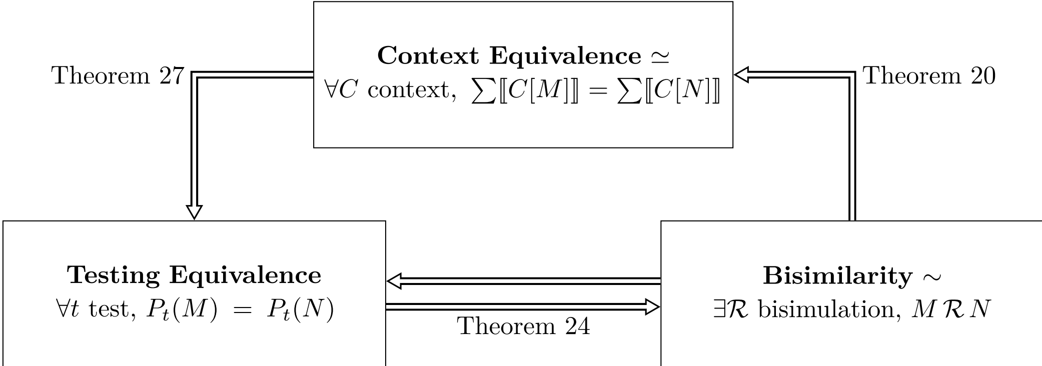


Figure 1 Sketch of the main results in the paper, giving Corollary 28.

The result is not surprising if compared to [3], however let us stress the contrast with the non-lazy cbn reduction strategy (i.e. the full head-reduction). We have already mentioned that [10] considers the cbn probabilistic PCF endowed with the let-in operator. The full abstraction result of probabilistic coherence spaces proved in [10] shows that the let-in operator does not change the context equivalence of probabilistic PCF, as this latter corresponds with the equality in probabilistic coherence spaces, regardless of the presence of the let-in in the language. Also, [2, 14] achieve a similar probabilistic coherence spaces full abstraction result for the untyped non-lazy cbn probabilistic λ-calculus without the let-in operator. These considerations show that the need of let-in operator for getting the full abstraction is due to the notion of *lazy* normal form rather than the call-by-name policy.

Structure of the paper. Section 2 defines $\Lambda_{\oplus,\mathsf{let}}$, the lazy call-by-name probabilistic λ-calculus extended with the let-in operator. The operational semantics is given by a notion of big-step approximation, following [8]. An equivalent notion based on Markov chains could be given as in e.g. [9]. The context equivalence is defined by Equation (5) where what we observe is the probability of getting a value. Notice that the notion of *lazyness* plays a crucial role here, since a value is a variable or just an abstraction and not a head-normal form, as it is the case instead in the non-lazy cbn considered in e.g. [2, 9, 13, 14].

Section 3 defines the probabilistic applicative bisimulation and the corresponding bisimilarity by considering $\Lambda_{\oplus,\mathsf{let}}$ as a labelled Markov chain. The definitions and results of this section are an adaptation of the ones in [7]. The main result is the soundness of bisimilarity with respect to the context equivalence (Theorem 20), whose proof is based on Lemma 19 stating that the bisimilarity is a congruence. The proof of this lemma is quite technical but follows the same lines of [3, 4, 7], using Howe's lifting: we postpone the details in the Appendix. The last Section 4 achieves the converse of Theorem 20 by considering Larsen and Skou's testing language (Definition 21) which is well-known to induce an equivalence corresponding with probabilistic bisimilarity (Theorem 24). Lemma 25 states that any test can be represented by a context of $\Lambda_{\oplus,\mathsf{let}}$ (here we are using in an essential way the presence of the let-in operator), so giving Theorem 27 and closing the circle (Corollary 28). Figure 1 sketches the main reasoning of the paper.

2 Preliminaries

In this section we introduce the syntax and operational semantics of $\Lambda_{\oplus,\mathsf{let}}$.

2.1 Probabilistic Lambda Calculus $\Lambda_{\oplus,\mathsf{let}}$

We present the probabilistic lambda calculus $\Lambda_{\oplus,\mathsf{let}}$, that is the pure, untyped lambda calculus endowed with two new operators: a probabilistic binary sum operator $\oplus$, representing a fair choice and a let-in operator, simulating the call-by-value evaluation in a call-by-name calculus. The operational semantics of $\Lambda_{\oplus,\mathsf{let}}$ is defined by a big-step approximation relation as in [8], we refer to this paper for more details. Given a countable set $X = \{x, y, z, \dots\}$ of variables, term expressions (*terms*) and *values* are generated by the following grammar:

$$\begin{array}{lll} \text{(values)} & V, W & ::= x \mid \lambda x.M, \\ \text{(terms)} & M, N & ::= V \mid MN \mid M \oplus N \mid \mathsf{let}\ x = M\ \mathsf{in}\ N, \end{array} \tag{1}$$

where $x \in X$. The set of all terms (resp. values) is denoted by $\Lambda_{\oplus,\mathsf{let}}$ (resp. $\mathcal{V}_{\oplus,\mathsf{let}}$) and is ranged over by capital Latin letters $M, N, \dots$, the letters V, W being reserved for values. The set of *free variables* of a term M is indicated as $\mathsf{FV}(M)$ and is defined in the usual way. Given a finite set of variables $\Gamma = \{x_1, \dots, x_n\} \subseteq X$, $\Lambda^\Gamma_{\oplus,\mathsf{let}}$ (resp. $\mathcal{V}^\Gamma_{\oplus,\mathsf{let}}$) denotes the set of terms (resp. values) whose free variables are within Γ. A term M is *closed* if $\mathsf{FV}(M) = \emptyset$, or equivalently if $M \in \Lambda^\emptyset_{\oplus,\mathsf{let}}$. The capture-avoiding substitution of N for the free occurrences of x in M is denoted by $M\{N/x\}$.

▶ **Example 1.** Let us define some terms useful in the sequel. The identity $\mathbf{I} \overset{\text{def}}{=} \lambda x.x$, the boolean projections $\mathbf{T} \overset{\text{def}}{=} \lambda xy.x$ and $\mathbf{F} \overset{\text{def}}{=} \lambda xy.y$ and the duplicator $\mathbf{\Delta} \overset{\text{def}}{=} \lambda x.xx$, this latter giving the ever looping term $\mathbf{\Omega} \overset{\text{def}}{=} \mathbf{\Delta\Delta}$. The let-in operator allows for a call-by-value duplicator $\mathbf{\Delta}^\ell \overset{\text{def}}{=} \lambda x.\mathsf{let}\ x = x\ \mathsf{in}\ xx$ that will distribute over the probabilistic choice (see Example 3).

Because of the probabilistic operator $\oplus$, a closed term does not evaluate to a single value, but to a discrete distribution of possible outcomes, i.e. to a function assigning a probability to any value. More formally, a *(value) distribution* is a map $\mathcal{D} : \mathcal{V}^\emptyset_{\oplus,\mathsf{let}} \to \mathbb{R}_{[0,1]}$ such that $\sum_{V \in \mathcal{V}^\emptyset_{\oplus,\mathsf{let}}} \mathcal{D}(V) \leq 1$. The set of all value distributions is denoted by $\mathcal{P}$. Given a value distribution $\mathcal{D}$, the set of all values to which $\mathcal{D}$ attributes a positive probability is denoted by $\mathsf{S}(\mathcal{D})$ and we will call it *the support of* $\mathcal{D}$. Note that value distributions do not necessarily sum to 1, this allowing to model the possibility of divergence (Example 5). We will use the abbreviation $\sum \mathcal{D}$ to stand for $\sum_{V \in \mathcal{V}^\emptyset_{\oplus,\mathsf{let}}} \mathcal{D}(V)$. The expression $p_1 V_1 + \cdots + p_n V_n$ denotes the distribution $\mathcal{D}$ with finite support $\{V_1, \dots, V_n\}$ such that $\mathcal{D}(V_i) = p_i$, for every $i \in \{1, \dots, n\}$. Note that $\sum \mathcal{D} = \sum_{i=1}^n p_i$. In particular, 0 denotes the empty distribution and V can denote both a value and the distribution having all of its mass on V.

The operational semantics of $\Lambda_{\oplus,\mathsf{let}}$ is given in two steps. First, the derivation rules in Figure 2 inductively define a notion of big-step approximation relation $M \Downarrow \mathcal{D}$ between a closed term M and a finite value distribution $\mathcal{D}$. Then, the semantics $[\![M]\!]$ of M is given as:

$$[\![M]\!] = \sup\{\mathcal{D}\ ;\ M \Downarrow \mathcal{D}\}, \tag{2}$$

according to the point-wise order over value distributions ($\mathcal{D} \leq \mathcal{E}$ if and only if $\forall V, \mathcal{D}(V) \leq \mathcal{E}(V)$). The lub in Equation (2) is well-defined since $\leq$ is a directed-complete partial order and the set $\{\mathcal{D}\ ;\ M \Downarrow \mathcal{D}\}$ is countable and directed (for every $M \Downarrow \mathcal{D}$ and $M \Downarrow \mathcal{E}$, there exists a distribution $\mathcal{F} \geq \mathcal{D}, \mathcal{E}$ such that $M \Downarrow \mathcal{F}$).

$$\frac{}{M \Downarrow 0} \qquad \frac{}{V \Downarrow V} \qquad \frac{M \Downarrow \mathcal{D} \quad N \Downarrow \mathcal{E}}{M \oplus N \Downarrow \frac{1}{2} \cdot \mathcal{D} + \frac{1}{2} \cdot \mathcal{E}}$$

$$\frac{M \Downarrow \mathcal{D} \quad \{P\{N/x\} \Downarrow \mathcal{E}_{P,N}\}_{\lambda x.P \in \mathsf{S}(\mathcal{D})}}{MN \Downarrow \sum_{\lambda x.P \in \mathsf{S}(\mathcal{D})} \mathcal{D}(\lambda x.P) \cdot \mathcal{E}_{P,N}} \qquad \frac{N \Downarrow \mathcal{G} \quad \{M\{V/x\} \Downarrow \mathcal{H}_V\}_{V \in \mathsf{S}(\mathcal{G})}}{\mathsf{let}\ x = N\ \mathsf{in}\ M \Downarrow \sum_{V \in \mathsf{S}(\mathcal{G})} \mathcal{G}(V) \cdot \mathcal{H}_V}$$

Figure 2 Rules for the approximation relation $M \Downarrow \mathcal{D}$, with $M \in \Lambda^{\emptyset}_{\oplus,\mathsf{let}}$ and $\mathcal{D}$ a value distribution.

$$\cfrac{V \Downarrow V \qquad \cfrac{\mathbf{I} \Downarrow \mathbf{I} \qquad \cfrac{V \Downarrow V \qquad \cfrac{\cfrac{\mathbf{I} \Downarrow \mathbf{I} \qquad VV \Downarrow 0}{\mathbf{I} \oplus VV \Downarrow \frac{1}{2}\mathbf{I}}}{\vdots}}{VV \Downarrow \sum_{i=1}^{n-1} \frac{1}{2^i}\mathbf{I}}}{\mathbf{I} \oplus VV \Downarrow \sum_{i=1}^{n} \frac{1}{2^i}\mathbf{I}}}{VV \Downarrow \sum_{i=1}^{n} \frac{1}{2^i}\mathbf{I}}$$

Figure 3 A derivation of the big-step approximation $VV \Downarrow \sum_{i=1}^{n} \frac{1}{2^i}\mathbf{I}$ for $V = \lambda x.(\mathbf{I} \oplus xx)$.

Notice that the rules in Figure 2 implement a lazy call-by-name evaluation: they do not reduce within the body of an abstraction, and an application $(\lambda x.M)N$ is evaluated as $M\{N/x\}$ for any term N. However, the let-in operator follows a call-by-value policy: let $x = N$ in M has the same semantics as $M\{N/x\}$ only when N is a value.

▶ **Example 2.** Consider the term $M \stackrel{\text{def}}{=} \mathbf{\Delta}(\mathbf{T} \oplus \mathbf{F})$. One can easily check that the rules of Figure 2 allows to derive $M \Downarrow \mathcal{D}$ for any $\mathcal{D} \in \{0, \frac{1}{2}\lambda y.(\mathbf{T} \oplus \mathbf{F}), \frac{1}{2}\mathbf{I}, \frac{1}{2}\lambda y.(\mathbf{T} \oplus \mathbf{F}) + \frac{1}{2}\mathbf{I}\}$. The latter distribution is the lub of this set and so it defines the semantics of M.

▶ **Example 3.** Let us replace in Example 2 the duplicator $\mathbf{\Delta}$ with its call-by-value variant $\mathbf{\Delta}^\ell$ (Example 1). We have $\mathbf{\Delta}^\ell(\mathbf{T} \oplus \mathbf{F}) \Downarrow \mathcal{D}$ for any $\mathcal{D} \in \{0, \frac{1}{2}\lambda y.\mathbf{T}, \frac{1}{2}\mathbf{I}, \frac{1}{2}\lambda y.\mathbf{T} + \frac{1}{2}\mathbf{I}\}$, so $[\![\mathbf{\Delta}^\ell(\mathbf{T} \oplus \mathbf{F})]\!] = \frac{1}{2}\lambda y.\mathbf{T} + \frac{1}{2}\mathbf{I}$. Notice that $[\![\mathbf{\Delta}^\ell(\mathbf{T} \oplus \mathbf{F})]\!] = [\![\mathbf{\Delta}^\ell\mathbf{T} \oplus \mathbf{\Delta}^\ell\mathbf{F}]\!] = [\![\mathbf{\Delta}\mathbf{T} \oplus \mathbf{\Delta}\mathbf{F}]\!]$, while $[\![\mathbf{\Delta}(\mathbf{T} \oplus \mathbf{F})]\!] \neq [\![\mathbf{\Delta}\mathbf{T} \oplus \mathbf{\Delta}\mathbf{F}]\!]$, as calculated in Example 2. Let us mention that this phenomenon is well enlightened by the linear logic encoding of the call-by-name application and the call-by-value one, the latter resulting in an operator linear both in the function and the argument position, while the former is linear only in the functional position [11].

▶ Remark 4. In general, notice that the presence of the let-in operator allows for an encoding $(\ )^\bullet$ of the call-by-value λ-calculus into our language, commuting with abstraction and variables and mapping the call-by-value application to $(MN)^\bullet = \mathsf{let}\ x = M^\bullet\ \mathsf{in}\ \mathsf{let}\ y = N^\bullet\ \mathsf{in}\ xy$. However, our language contains much more terms than the image of this mapping, so the computational behaviour of the terms might not be preserved by $(\ )^\bullet$.

▶ **Example 5.** The previous examples are about normalizing terms, in this framework meaning terms M with semantics of total mass $\sum[\![M]\!] = 1$ and such that there exists a unique finite derivation giving $M \Downarrow [\![M]\!]$. Standard non-converging λ-terms give partiality, as for example $[\![\mathbf{\Omega}]\!] = 0$, so $[\![\mathbf{\Omega} \oplus \mathbf{I}]\!] = \frac{1}{2}\mathbf{I}$. However, probabilistic λ-calculi allow for almost sure terminating terms, that is terms M such that $\sum[\![M]\!] = 1$ but there exists no finite derivation giving $M \Downarrow [\![M]\!]$. For example, consider the term $M \stackrel{\text{def}}{=} VV$, with $V \stackrel{\text{def}}{=} \lambda x.(\mathbf{I} \oplus xx)$: any finite approximation of M gives a distribution bounded by $\sum_{i=1}^{n} \frac{1}{2^i}\mathbf{I}$ for some $n \geq 0$, as Figure 3 shows, but only the limit sum $\sup_n \sum_{i=1}^{n} \frac{1}{2^i}\mathbf{I}$ is equal to $[\![M]\!] = \mathbf{I}$.

The following lemma states simple properties of the semantics that can be easily proved by continuity of $[\![\ ]\!]$ and induction over finite approximations (see e.g. [8] for details).

▶ **Lemma 6** ([8]). *For any terms M and N,*
1. $[\![(\lambda x.M)N]\!] = [\![M\{N/x\}]\!]$.
2. $[\![M \oplus N]\!] = \frac{1}{2}[\![M]\!] + \frac{1}{2}[\![N]\!]$.

2.2 Context Equivalence

One standard way of comparing term expressions is by observing their behaviours within programming contexts. A *context* of $\Lambda_{\oplus,\mathsf{let}}$ is a term containing a unique hole $[\cdot]$, generated by the following grammar:

$$C, D ::= [\cdot] \mid \lambda x.C \mid CM \mid MC \mid C \oplus M \mid M \oplus C \mid \mathsf{let}\ x = C\ \mathsf{in}\ M \mid \mathsf{let}\ x = M\ \mathsf{in}\ C \tag{3}$$

If C is a context and M is a $\Lambda_{\oplus,\mathsf{let}}$-term, then $C[M]$ denotes a $\Lambda_{\oplus,\mathsf{let}}$-term obtained by substituting the unique hole in C with M allowing the possible capture of free variables of M. We will work with closing contexts, that is contexts C such that $C[M]$ is a closed term (where M can be an open term). Thus, we want to keep track of the possible variables captured by filling a context hole. Given two finite sets of variables Γ, Δ, we denote by $\mathsf{C}\Lambda_{\oplus,\mathsf{let}}{}^{(\Gamma;\Delta)}$ the set of contexts capturing the variables in Γ of a term filling the hole but keeping free the variables in Δ. So for example the context $\lambda x.\mathsf{let}\ y = x \oplus z\ \mathsf{in}\ x[\cdot]$ belongs to $\mathsf{C}\Lambda_{\oplus,\mathsf{let}}{}^{(\{x,y\};\Delta)}$ for any Δ containing z.

In a probabilistic setting, the typical observation is the probability to converge to a value, so giving the following standard definition, for every $M, N \in \Lambda^{\Gamma}_{\oplus,\mathsf{let}}$:

$$M \leq N \text{ iff } \forall C \in \mathsf{C}\Lambda_{\oplus,\mathsf{let}}{}^{(\Gamma;\emptyset)}, \sum [\![C[M]]\!] \leq \sum [\![C[N]]\!], \qquad \text{(context preorder)} \tag{4}$$

$$M \simeq N \text{ iff } \forall C \in \mathsf{C}\Lambda_{\oplus,\mathsf{let}}{}^{(\Gamma;\emptyset)}, \sum [\![C[M]]\!] = \sum [\![C[N]]\!] \qquad \text{(context equivalence)} \tag{5}$$

Notice that $M \simeq N$ is equivalent to $M \leq N$ and $N \leq M$.

▶ **Example 7.** As mentioned in the Introduction, the terms $M \stackrel{\text{def}}{=} \lambda xy.(x \oplus y)$ and $N \stackrel{\text{def}}{=} (\lambda xy.x) \oplus (\lambda xy.y)$ are context equivalent in the call-by-name probabilistic λ-calculus without the let-in operator [7]. However, they can be discriminated in $\Lambda_{\oplus,\mathsf{let}}$ by, e.g. the context $C \stackrel{\text{def}}{=} (\mathsf{let}\ y = [\cdot]\ \mathsf{in}\ (\mathsf{let}\ z_1 = y\mathbf{I}\mathbf{\Omega}\ \mathsf{in}\ (\mathsf{let}\ z_2 = y\mathbf{I}\mathbf{\Omega}\ \mathsf{in}\ \mathbf{I})))$. In fact, by applying the rules of Figure 2, one gets: $\sum [\![C[M]]\!] = \frac{1}{4}$ and $\sum [\![C[N]]\!] = \frac{1}{2}$.

▶ **Example 8.** The two duplicators $\mathbf{\Delta}$ and $\mathbf{\Delta}^\ell$ (Example 1) are not context equivalent, for example $C \stackrel{\text{def}}{=} [\cdot](\mathbf{I} \oplus \mathbf{\Omega})$ gives $\sum [\![C[\mathbf{\Delta}]]\!] = \frac{1}{4}$ while $\sum [\![C[\mathbf{\Delta}^\ell]]\!] = \frac{1}{2}$.

▶ **Proposition 9.** *Let $M, N \in \Lambda^{\emptyset}_{\oplus,\mathsf{let}}$, if $[\![M]\!] \leq [\![N]\!]$ then $M \leq N$. So, $[\![M]\!] = [\![N]\!]$ implies $M \simeq N$.*

Proof. First, notice that $[\![M]\!] \leq [\![N]\!]$ is equivalent to $\forall \mathcal{D}, M \Downarrow \mathcal{D}, \exists \mathcal{E} \geq \mathcal{D}, N \Downarrow \mathcal{E}$. Then one proves, by structural induction on a context C that $[\![C(M)]\!] \leq [\![C(N)]\!]$, whenever $[\![M]\!] \leq [\![N]\!]$. The delicate points are in the cases C is an application or a let-in operator. ◀

▶ **Example 10.** Thanks to Proposition 9, one can prove that quite different terms are indeed context equivalent, e.g. the term VV in Example 5 is context equivalent to $\mathbf{I}$. However, not all context equivalent terms have the same semantics, as for example $\lambda x.(x \oplus x)$ and $\mathbf{I}$.

Proving in general that two terms are context equivalent is rather difficult because of the universal quantifier in Equation (5). For example, proving that $\lambda x.(x \oplus x)$ and $\mathbf{I}$ are context equivalent is not immediate. Various other tools are then used to prove context equivalence, as the bisimilarity and testing introduced in the next sections.

3 Probabilistic Applicative Bisimulation

We briefly recall and adapt to $\Lambda_{\oplus,\mathsf{let}}$ the definitions of [7] about probabilistic applicative (bi)simulation. This notion mixes Larsen and Skou's definition of (bi)simulation for labelled Markov chains [12] with Abramsky's applicative (bi)simulation for the lazy call-by-name λ-calculus [1]. The core idea is to look at a closed term M as a state of a transition system, a Markov chain in our setting, having two kinds of transitions. A "solipsistic" transition consisting in evaluating M to a value $\lambda x.P$ (this transition being weighted by the probability $[\![M]\!](\lambda x.P)$ of getting $\lambda x.P$ out of M) and an "interactive" transition consisting in feeding a value $\lambda x.P$ by a new term N representing an input from the environment, so getting the term $P\{N/x\}$. We can then consider the notions of similarity and bisimilarity (resp. (6), (7)) over such probabilistic transition system. The benefit of this approach is to check program equivalence via an existential quantifier (see Equation (7)) rather than a universal one as in context equivalence (Equation (5)). The main result of this section is Theorem 20 stating that similarity implies context preorder. As a consequence we have that bisimilarity implies context equivalence. The key ingredient for achieving this result is to show that the similarity is a precongruence relation (Definition 18 and Lemma 19). The proof of Lemma 19 is quite technical but standard, see the Appendix and [7] for more details.

We start with the definition of a generic labelled Markov chain and following Larsen and Skou [12] we introduce the notions of a probabilistic simulation and bisimulation.

▶ **Definition 11.** *A* labelled Markov chain *is a triple $\mathcal{M} = (\mathcal{S}, \mathcal{L}, P)$ where $\mathcal{S}$ is a countable set of states, $\mathcal{L}$ is a set of labels (actions) and P is a transition probability matrix, i.e. a function $P : \mathcal{S} \times \mathcal{L} \times \mathcal{S} \to \mathbb{R}_{[0,1]}$ satisfying the following condition: $\forall s \in \mathcal{S}, \forall l \in \mathcal{L}, \sum_{t\in\mathcal{S}} P(s, l, t) \leq 1$.*

Given a relation $\mathcal{R}$, $\mathcal{R}(X)$ denotes the image of the set X under $\mathcal{R}$, namely the set $\{y \mid \exists x \in X \text{ such that } x\mathcal{R}y\}$. If $\mathcal{R}$ is an equivalence relation, then $\mathcal{S}/\mathcal{R}$ stands for the set of all equivalence classes of $\mathcal{S}$ modulo $\mathcal{R}$. The expression $P(s, l, X)$ stands for $\sum_{t\in X} P(s, l, t)$.

▶ **Definition 12.** *Let $(\mathcal{S}, \mathcal{L}, P)$ be a labelled Markov chain and $\mathcal{R}$ be a relation over $\mathcal{S}$:*

- *$\mathcal{R}$ is a* probabilistic simulation *if it is a preorder and $\forall(s, t) \in \mathcal{R}, \forall X \subseteq \mathcal{S}, \forall l \in \mathcal{L}$, $P(s, l, X) \leq P(t, l, \mathcal{R}(X))$.*
- *$\mathcal{R}$ is a* probabilistic bisimulation *if it is an equivalence and $\forall(s, t) \in \mathcal{R}, \forall E \in \mathcal{S}/\mathcal{R}, \forall l \in \mathcal{L}$, $P(s, l, E) = P(t, l, E)$.*

We define the probabilistic (bi)similarity, denoted respectively by $\lesssim$ and $\simeq$, as the union of all probabilistic (bi)simulations which can be proven to be still a (bi)simulation:

$$M \lesssim N \text{ iff } \exists\mathcal{R} \text{ probabilistic simulation s.t. } M\mathcal{R}N, \qquad \text{(probabilistic similarity)} \quad (6)$$

$$M \sim N \text{ iff } \exists\mathcal{R} \text{ probabilistic bisimulation s.t. } M\mathcal{R}N \qquad \text{(probabilistic bisimilarity)} \quad (7)$$

One can prove that $M \sim N$ is equivalent to $M \lesssim N$ and $N \lesssim M$, i.e. $\sim = \lesssim \cap \lesssim^{op}$.

▶ **Definition 13.** *For every closed value $V = \lambda x.N \in \Lambda^{\emptyset}_{\oplus,\mathsf{let}}$ a distinguished value is indicated as $\widetilde{V} = \nu x.N$ and belongs to the set $\mathsf{V}\Lambda^{\emptyset}_{\oplus,\mathsf{let}}$.*

As an example, value $\lambda xy.x$ belongs to the set $\Lambda^{\emptyset}_{\oplus,\mathsf{let}}$, while the distinguished value $\nu x.\lambda y.x$ is the element of $\mathsf{V}\Lambda^{\emptyset}_{\oplus,\mathsf{let}}$. As previously stated, we want to see the operational semantics of $\Lambda_{\oplus,\mathsf{let}}$ as a labelled Markov chain defined as follows:

▶ **Definition 14.** *The $\Lambda_{\oplus,\mathsf{let}}$-Markov chain is defined as the triple $(\Lambda^{\emptyset}_{\oplus,\mathsf{let}} \uplus \mathsf{V}\Lambda^{\emptyset}_{\oplus,\mathsf{let}}, \Lambda^{\emptyset}_{\oplus,\mathsf{let}} \cup \{\tau\}, P)$, where the set of states is the disjoint union of the set of closed terms and the set of closed distinguished values, labels (actions) are either closed terms or τ action and the transition probability matrix P is defined in the following way:*

- *for every closed term M and distinguished value $\nu x.N$,*

$$P(M, \tau, \nu x.N) = [\![M]\!](\lambda x.N)\,,$$

- *for every closed term M and distinguished value $\nu x.N$,*

$$P(\nu x.N, M, N\{M/x\}) = 1\,,$$

- *in all other cases, P returns 0.*

For technical reasons the set of states is represented as a disjoint union $\Lambda^{\emptyset}_{\oplus,\mathsf{let}} \uplus \mathsf{V}\Lambda^{\emptyset}_{\oplus,\mathsf{let}}$.

Since $\Lambda_{\oplus,\mathsf{let}}$ can be seen as a labelled Markov chain, the simulation and bisimulation can be defined as for any labelled Markov chain. *A probabilistic applicative simulation* is a probabilistic simulation on $\Lambda_{\oplus,\mathsf{let}}$ and *a probabilistic applicative bisimulation* is a probabilistic bisimulation on $\Lambda_{\oplus,\mathsf{let}}$. Then, the *probabilistic applicative similarity*, PAS for short, and the *probabilistic applicative bisimilarity*, PAB for short, are defined in the usual way applying Equation (6) and (7). From now on, the symbol $\lesssim$ (resp. $\sim$) will denote the probabilistic applicative similarity (resp. bisimilarity).

The notions of PAS and PAB are defined on closed terms, and we extend these definitions to open terms by requiring the usual closure under substitutions. Let $M, N \in \Lambda^{\Gamma}_{\oplus,\mathsf{let}}$ where $\Gamma = \{x_1, \ldots, x_n\}$. We say M and N are similar, (denoted $M \lesssim N$), if for all $L_1 \in \Lambda^{\emptyset}_{\oplus,\mathsf{let}}, \ldots, L_n \in \Lambda^{\emptyset}_{\oplus,\mathsf{let}}$, $M\{L_1/x_1, \ldots, L_n/x_n\} \lesssim N\{L_1/x_1, \ldots, L_n/x_n\}$. The analogous terminology is introduced for bisimilarity.

▶ **Example 15.** Let us recall the terms $\lambda x.(x \oplus x)$ and $\lambda x.x$ from Example 10 having different semantics but context equivalent. As mentioned, the proof of their context equivalence is not immediate, because of the universal quantifier in Equation (5). However, we can check easily that they are bisimilar, because we need just to exhibit a bisimulation relation between the two terms. By Theorem 20 we then infer context equivalence from bisimilarity. Let us define the relation $\mathcal{R} = \{(\lambda x.(x \oplus x), \lambda x.x)\} \cup \{(\lambda x.x, \lambda x.(x \oplus x))\} \cup \{(\nu x.(x \oplus x), \nu x.x)\} \cup \{(\nu x.x, \nu x.(x \oplus x))\} \cup \{(N \oplus N, N) \mid N \in \Lambda^{\emptyset}_{\oplus,\mathsf{let}}\} \cup \{(N, N \oplus N) \mid N \in \Lambda^{\emptyset}_{\oplus,\mathsf{let}}\} \cup \{(M, M) \mid M \in \Lambda^{\emptyset}_{\oplus,\mathsf{let}}\} \cup \{(\widetilde{V}, \widetilde{V}) \mid \widetilde{V} \in \mathsf{V}\Lambda^{\emptyset}_{\oplus,\mathsf{let}}\}$. We prove that $\mathcal{R}$ is a bisimulation containing $(\lambda x.(x \oplus x), \lambda x.x)$. The relation is trivially an equivalence, so we have to show that $\forall (M, N) \in \mathcal{R}, \forall E \in (\Lambda^{\emptyset}_{\oplus,\mathsf{let}} \uplus \mathsf{V}\Lambda^{\emptyset}_{\oplus,\mathsf{let}})/\mathcal{R}, \forall \ell \in \Lambda^{\emptyset}_{\oplus,\mathsf{let}} \cup \{\tau\}$, $P(M, \ell, E) = P(N, \ell, E)$ (Definition 12). We prove only for $(\lambda x.(x \oplus x), \lambda x.x) \in \mathcal{R}$, $(\nu x.(x \oplus x), \nu x.x) \in \mathcal{R}$ and $(N \oplus N, N) \in \mathcal{R}$. First we have that $(\lambda x.(x \oplus x), \lambda x.x) \in \mathcal{R}$ and for all closed terms $F \in \Lambda^{\emptyset}_{\oplus,\mathsf{let}}$ and all equivalence classes $E \in (\Lambda^{\emptyset}_{\oplus,\mathsf{let}} \uplus \mathsf{V}\Lambda^{\emptyset}_{\oplus,\mathsf{let}})/\mathcal{R}$, $P(\lambda x.(x \oplus x), F, E) = 0 = P(\lambda x.x, F, E)$ holds by Definition 14. If the equivalence class E contains $\nu x.(x \oplus x)$ then $P(\lambda x.(x \oplus x), \tau, E) = 1$, otherwise $P(\lambda x.(x \oplus x), \tau, E) = 0$. Since $(\nu x.(x \oplus x), \nu x.x) \in \mathcal{R}$, we have that $\nu x.(x \oplus x) \in E$ if and only if $\nu x.x \in E$. Hence, $P(\lambda x.(x \oplus x), \ell, E) = P(\lambda x.x, \ell, E)$ for all $\ell \in \Lambda^{\emptyset}_{\oplus,\mathsf{let}} \cup \{\tau\}$ and all $E \in (\Lambda^{\emptyset}_{\oplus,\mathsf{let}} \uplus \mathsf{V}\Lambda^{\emptyset}_{\oplus,\mathsf{let}})/\mathcal{R}$. For all equivalence classes $E \in (\Lambda^{\emptyset}_{\oplus,\mathsf{let}} \uplus \mathsf{V}\Lambda^{\emptyset}_{\oplus,\mathsf{let}})/\mathcal{R}$, $P(\nu x.(x \oplus x), \tau, E) = 0 = P(\nu x.x, \tau, E)$ holds by Definition 14. Further, $P(\nu x.(x \oplus x), F, E) = 1$ for some $F \in \Lambda^{\emptyset}_{\oplus,\mathsf{let}}$ if $F \oplus F \in E$, otherwise $P(\nu x.(x \oplus x), F, E) = 0$. We have that $F \oplus F \in E$ if and only if $F \in E$, because $(F \oplus F, F) \in \mathcal{R}$ for all $F \in \Lambda^{\emptyset}_{\oplus,\mathsf{let}}$. Hence, $P(\nu x.(x \oplus x), \ell, E) = P(\nu x.x, \ell, E)$ for all $\ell \in \Lambda^{\emptyset}_{\oplus,\mathsf{let}} \cup \{\tau\}$ and all $E \in (\Lambda^{\emptyset}_{\oplus,\mathsf{let}} \uplus \mathsf{V}\Lambda^{\emptyset}_{\oplus,\mathsf{let}})/\mathcal{R}$. Finally, let us consider $(N \oplus N, N) \in \mathcal{R}$, for an arbitrary $N \in \Lambda^{\emptyset}_{\oplus,\mathsf{let}}$. For all closed terms $F \in \Lambda^{\emptyset}_{\oplus,\mathsf{let}}$ and all equivalence classes

$E \in (\Lambda^{\emptyset}_{\oplus,\mathsf{let}} \uplus \mathsf{V}\Lambda^{\emptyset}_{\oplus,\mathsf{let}})/\mathcal{R}$, $P(N \oplus N, F, E) = 0 = P(N, F, E)$ holds by Definition 14. By Lemma 6 and Definition 14 we have that the following holds for all equivalence classes $E \in (\Lambda^{\emptyset}_{\oplus,\mathsf{let}} \uplus \mathsf{V}\Lambda^{\emptyset}_{\oplus,\mathsf{let}})/\mathcal{R}$.

$$\begin{aligned} P(N \oplus N, \tau, E) &= \sum_{\nu x.M \in E} P(N \oplus N, \tau, \nu x.M) = \sum_{\{\lambda x.M \mid \nu x.M \in E\}} [\![N \oplus N]\!](\lambda x.M) \\ &= \sum_{\{\lambda x.M \mid \nu x.M \in E\}} (\frac{1}{2}[\![N]\!] + \frac{1}{2}[\![N]\!])(\lambda x.M) = \sum_{\{\lambda x.M \mid \nu x.M \in E\}} [\![N]\!](\lambda x.M) \\ &= \sum_{\nu x.M \in E} P(N, \tau, \nu x.M) = P(N, \tau, E) \end{aligned}$$

The proof for the other elements of $\mathcal{R}$ is analogous to the cases we considered.

▶ **Example 16.** The terms $M = \lambda xy.(x \oplus y)$ and $N = (\lambda xy.x) \oplus (\lambda xy.y)$ are not bisimilar. Let us suppose the opposite. Then, there exists a bisimulation $\mathcal{R}$ such that $(M, N) \in \mathcal{R}$. By definition $\mathcal{R}$ is an equivalence relation. Let E be an equivalence class of $\Lambda^{\emptyset}_{\oplus,\mathsf{let}} \uplus \mathsf{V}\Lambda^{\emptyset}_{\oplus,\mathsf{let}}$ with respect to $\mathcal{R}$ which contains $\nu x.\lambda y.(x \oplus y)$. Then, we should have that $1 = P(M, \tau, E) = P(N, \tau, E)$. We know that $P(N, \tau, \nu x.\lambda y.x) = \frac{1}{2}$ and $P(N, \tau, \nu x.\lambda y.y) = \frac{1}{2}$. Thus, we can conclude $\nu x.\lambda y.x \in E$ and $\nu x.\lambda y.y \in E$. If $\nu x.\lambda y.x \in E$, then $(\nu x.\lambda y.(x \oplus y), \nu x.\lambda y.x) \in \mathcal{R}$. Hence we have that $1 = P(\nu x.\lambda y.(x \oplus y), \mathbf{\Omega}, E_1) = P(\nu x.\lambda y.x, \mathbf{\Omega}, E_1)$, where E_1 is an equivalence class which contains $\lambda y.(\mathbf{\Omega} \oplus y)$. Using the fact that $P(\nu x.\lambda y.x, \mathbf{\Omega}, \lambda y.\mathbf{\Omega}) = 1$ we obtain $\lambda y.\mathbf{\Omega} \in E_1$. Since $\lambda y.(\mathbf{\Omega} \oplus y)$ and $\lambda y.\mathbf{\Omega}$ belong to the same equivalence class we conclude $(\lambda y.(\mathbf{\Omega} \oplus y), \lambda y.\mathbf{\Omega}) \in \mathcal{R}$. If E_2 is an equivalence class such that $\nu y.(\mathbf{\Omega} \oplus y) \in E_2$, then we have that $1 = P(\lambda y.(\mathbf{\Omega} \oplus y), \tau, E_2) = P(\lambda y.\mathbf{\Omega}, \tau, E_2)$. By a similar reasoning as before we obtain that $(\nu y.(\mathbf{\Omega} \oplus y), \nu y.\mathbf{\Omega}) \in \mathcal{R}$. Let E_3 be an equivalence class which contains $\mathbf{\Omega} \oplus \mathbf{I}$. From $1 = P(\nu y.(\mathbf{\Omega} \oplus y), \mathbf{I}, E_3) = P(\nu y.\mathbf{\Omega}, \mathbf{I}, E_3)$ it follows that $\mathbf{\Omega} \in E_3$, i.e. $(\mathbf{\Omega} \oplus \mathbf{I}, \mathbf{\Omega}) \in \mathcal{R}$. Finally, if E_4 is an equivalence class such that $\nu x.x \in E_4$, then $\frac{1}{2} = P(\mathbf{\Omega} \oplus \mathbf{I}, \tau, E_4) = P(\mathbf{\Omega}, \tau, E_4)$. This is in contradiction with $P(\mathbf{\Omega}, \tau, E_4) = 0$ which is a consequence of the definition of a transition probability matrix. Thus, terms M and N are not bisimilar.

The following proposition is the analogous to Proposition 9, stating the soundness of (bi)simulation with respect to the operational semantics.

▶ **Proposition 17.** *Let $M, N \in \Lambda^{\emptyset}_{\oplus,\mathsf{let}}$, if $[\![M]\!] \leq [\![N]\!]$ then $M \lesssim N$. So, $[\![M]\!] = [\![N]\!]$ implies $M \sim N$.*

Proof. By checking that the relation $\mathcal{R} = \{(M, N) \in \Lambda^{\emptyset}_{\oplus,\mathsf{let}} \times \Lambda^{\emptyset}_{\oplus,\mathsf{let}} \mid [\![M]\!] \leq [\![N]\!]\} \cup \{(\widetilde{V}, \widetilde{V}) \in \mathsf{V}\Lambda^{\emptyset}_{\oplus,\mathsf{let}} \times \mathsf{V}\Lambda^{\emptyset}_{\oplus,\mathsf{let}}\}$ is a probabilistic applicative simulation. The second part of the statement follows from $\sim = \lesssim \cap (\lesssim)^{op}$. ◀

We introduce a new notion of relations called $\Lambda_{\oplus,\mathsf{let}}$-relations, which are sets of triples in the form (Γ, M, N) where $M, N \in \Lambda^{\Gamma}_{\oplus,\mathsf{let}}$. Any relation R' on the set of $\Lambda_{\oplus,\mathsf{let}}$-terms can be extended to a $\Lambda_{\oplus,\mathsf{let}}$-relation $\mathcal{R}$, such that whenever $(M, N) \in R'$ and $M, N \in \Lambda^{\Gamma}_{\oplus,\mathsf{let}}$, we have that $(\Gamma, M, N) \in \mathcal{R}$. We will write $\Gamma \vdash M\mathcal{R}N$ instead of $(\Gamma, M, N) \in \mathcal{R}$.

▶ **Definition 18.** *A $\Lambda_{\oplus,\mathsf{let}}$-relation $\mathcal{R}$ is a* congruence *(respectively,* a precongruence*) if it is an equivalence (respectively, a preorder) and for every $\Gamma \cup \Delta \vdash M\mathcal{R}N$ and every context $C \in \mathsf{C}\Lambda_{\oplus,\mathsf{let}}{}^{(\Gamma;\Delta)}$, we have that $\Delta \vdash C[M]\mathcal{R}C[N]$.*

It is immediate to check that the context preorder $\leq$ (resp. equivalence $\simeq$) is a precongruence (resp. congruence)(Appendix A.1). Also similarity is a precongruence, but its proof is more involved (Appendix A.2). As a consequence we have that bisimilarity is a congruence.

▶ **Lemma 19.** *The similarity $\lesssim$ (resp. bisimilarity $\sim$) is a precongruence (resp. congruence) relation for $\Lambda_{\oplus,\mathsf{let}}$-terms.*

Proof (Sketch). As standard [3, 4, 7], we use Howe's technique to prove that probabilistic similarity is a precongruence, this implying that the probabilistic bisimilarity is also a congruence. The proof is technical and follows the same reasoning as [7], the only difference being in the cases needed to handle the compatibility associated with the let-in operator.

We start with defining Howe's lifting for $\Lambda_{\oplus,\mathsf{let}}$, which turns an arbitrary relation $\mathcal{R}$ to another one $\mathcal{R}^H$. The relation $\mathcal{R}^H$ enjoys some properties with respect to the relation $\mathcal{R}$. In particular, if $\mathcal{R}$ is reflexive, transitive and closed under term-substitution, then it is included in $\mathcal{R}^H$ and the relation $\mathcal{R}^H$ is context closed and also closed under term-substitution. These properties allow to prove that the transitive closure $(\lesssim^H)^+$ of the Howe's lifting $\lesssim^H$ is a precongruence including $\lesssim$. One can conclude then easily that $\lesssim$ is also a precongruence. Finally, from $\sim=\lesssim \cup (\lesssim)^{op}$ we conclude that $\sim$ is a congruence. ◀

Now we can prove that simulation preorder (similarity) is sound with respect to the context preorder. As a consequence we have that bisimulation equivalence (bisimilarity) is included in the context equivalence.

▶ **Theorem 20** (Soundness). *For every $M, N \in \Lambda^{\Gamma}_{\oplus,\mathsf{let}}$, $\Gamma \vdash M \lesssim N$ implies $\Gamma \vdash M \leq N$. Therefore, $M \sim N$ implies $\Gamma \vdash M \simeq N$.*

Proof. Suppose that $\Gamma \vdash M \lesssim N$. We have that for every context $C \in \mathsf{C}\Lambda_{\oplus,\mathsf{let}}^{(\Gamma;\emptyset)}$, $\emptyset \vdash C[M] \lesssim C[N]$ holds as a consequence of Lemma 19. Then by definition there exists a simulation between $C[M]$ and $C[N]$, which implies by Definition 12 that $\sum[\![C[M]]\!] \leq \sum[\![C[N]]\!]$ holds. We conclude $\Gamma \vdash M \leq N$. The second part of the statement follows from the definitions $\sim=\lesssim \cap \lesssim^{op}$ and $\simeq=\leq \cap \leq^{op}$. ◀

4 Full Abstraction

The goal of this section is to prove the converse of Theorem 20, showing that context equivalence and bisimilarity coincide. In order to get this result, it is more convenient to use the notion of testing equivalence, which has been proven to coincide with Markov processes bisimilarity in [16] (here Theorem 24). In this framework we need to consider only Markov chains, which are the discrete-time version of Markov processes, so we simplify the definitions and results of [16] to this discrete setting, following [3]. Notice that Theorem 24 is independent from the particular Markov chain considered, so we recall the general definitions and then we applied them to the $\Lambda_{\oplus,\mathsf{let}}$-Markov chain.

▶ **Definition 21** ([3]). *Let $(\mathcal{S}, \mathcal{L}, \mathcal{P})$ be a labelled Markov chain. The testing language $\mathcal{T}_{(\mathcal{S},\mathcal{L},\mathcal{P})}$ for $(\mathcal{S}, \mathcal{L}, \mathcal{P})$ is given by the grammar*

$$t ::= \omega \mid a.t \mid (t, t),$$

where ω is a symbol for termination and $a \in \mathcal{L}$ is an action (label).

It is easy to see that tests are finite objects. A test is an algorithm for doing an experiment on a program. During the execution of a test on a particular program, one can observe the success or the failure of the experiment with a given probability. The symbol ω represents a test which does not require an experiment at all (it always succeed). The test $a.t$ describes an experiment consisting of performing the action a and in the case of success performing the test t, and the test (t, s) makes two copies of the current state and allows both tests t and s to be performed independently on the same state. The success probability of a test is defined as follows:

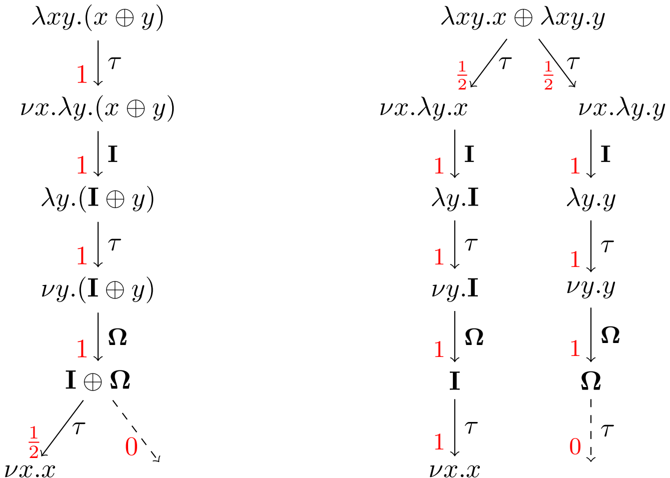


Figure 4 The experiment $t = \tau.(\mathbf{I}.\tau.\mathbf{\Omega}.\tau.\omega, \mathbf{I}.\tau.\mathbf{\Omega}.\tau.\omega)$ over the terms of Example 23.

▶ **Definition 22** ([3]). *Let $(\mathcal{S}, \mathcal{L}, \mathcal{P})$ be a labelled Markov chain. We define a family $\{P_t(\cdot)\}_{t \in \mathcal{T}_{(\mathcal{S},\mathcal{L},\mathcal{P})}}$ of maps from the set of states $\mathcal{S}$ to $\mathbb{R}_{[0,1]}$, by induction on the structure of t:*

- $P_\omega(s) = 1$;
- $P_{a.t}(s) = \sum_{s' \in \mathcal{S}} \mathcal{P}(s, a, s') P_t(s')$;
- $P_{(t_1,t_2)}(s) = P_{t_1}(s) \cdot P_{t_2}(s)$.

▶ **Example 23.** The terms $\lambda xy.(x \oplus y)$ and $(\lambda xy.x) \oplus (\lambda xy.y)$ of Example 7 can be discriminated by the test $t = \tau.(\mathbf{I}.\tau.\mathbf{\Omega}.\tau.\omega, \mathbf{I}.\tau.\mathbf{\Omega}.\tau.\omega)$. Figure 4 sketches the computation of $P_t(\lambda xy.(x \oplus y)) = \frac{1}{4}$ and $P_t((\lambda xy.x) \oplus (\lambda xy.y)) = \frac{1}{2}$.

The following theorem states the equivalence between the notion of bisimilarity over $(\mathcal{S}, \mathcal{L}, \mathcal{P})$ and testing equivalence. The theorem has been proven in [16] for a labelled Markov processes. For lack of space, we have omitted a detailed proof of the adaptation of the results from labelled Markov processes to labelled Markov chains.

▶ **Theorem 24** ([3],[16]). *Let $(\mathcal{S}, \mathcal{L}, \mathcal{P})$ be a labelled Markov chain. Then $s, s' \in \mathcal{S}$ are bisimilar if and only if $P_t(s) = P_t(s')$ for every test $t \in \mathcal{T}_{(\mathcal{S},\mathcal{L},\mathcal{P})}$.*

It is known that this theorem does not hold for inequalities [16]. More precisely, it is not true that $s \lesssim s'$ just in case $P_t(s) \leq P_t(s')$ for every test $t \in \mathcal{T}_{(\mathcal{S},\mathcal{L},\mathcal{P})}$.

4.1 Every Test has an Equivalent Context

Here is the main contribution of our paper, showing that for every test t associated with the $\Lambda_{\oplus,\mathsf{let}}$-Markov chain there exists a context C_t expressing t in the syntax of $\Lambda_{\oplus,\mathsf{let}}$, i.e. $P_t(M) = \sum [\![C_t[M]]\!]$ for every term M (Lemma 25). So context equivalence implies testing equivalence (Theorem 27) and hence bisimilarity by Theorem 24. Together with Theorem 20 this achieves the diagram in Figure 1, so Corollary 28.

▶ **Lemma 25.** *For every test $t \in \mathcal{T}_{\Lambda_{\oplus,\mathsf{let}}}$, there are contexts $C_t \in \mathsf{C}\Lambda_{\oplus,\mathsf{let}}{}^{(\emptyset;\emptyset)}$ and $D_t \in \mathsf{C}\Lambda_{\oplus,\mathsf{let}}{}^{(\emptyset;\emptyset)}$ such that for every term $M \in \Lambda^{\emptyset}_{\oplus,\mathsf{let}}$ and value $V = \lambda x.M \in \mathcal{V}^{\emptyset}_{\oplus,\mathsf{let}}$ it holds that:*

$$P_t(M) = \sum [\![C_t[M]]\!] \quad \textit{and} \quad P_t(\widetilde{V}) = \sum [\![D_t[V]]\!],$$

where we recall that $\widetilde{V}$ denotes the distinguished value $\nu x.M \in \mathsf{V}\Lambda^{\emptyset}_{\oplus,\mathsf{let}}$.

Proof. We prove it by induction on the structure of a test t.

- First we consider the case where $t = \omega$. Then, by the definition of $P_t(\cdot)$, we have that for every $M \in \Lambda^{\emptyset}_{\oplus,\mathsf{let}}$ and $V \in \mathcal{V}^{\emptyset}_{\oplus,\mathsf{let}}$, $P_\omega(M) = 1$ and $P_\omega(\widetilde{V}) = 1$. Thus, we can define $C_\omega = (\lambda xy.x)[\cdot]$ and $D_\omega = (\lambda xy.x)[\cdot]$ and we obtain, for every $M \in \Lambda^{\emptyset}_{\oplus,\mathsf{let}}$

$$\sum [\![C_\omega[M]]\!] = \sum [\![(\lambda xy.x)M]\!] = \sum [\![\lambda y.M]\!] = 1 = P_\omega(M),$$

and for every value $V \in \mathcal{V}^{\emptyset}_{\oplus,\mathsf{let}}$

$$\sum [\![D_\omega[V]]\!] = \sum [\![(\lambda xy.x)V]\!] = \sum [\![\lambda y.V]\!] = 1 = P_\omega(\widetilde{V}).$$

- Next, let us consider the case where $t = a.t'$ for some action (label) a. By induction hypothesis there are contexts $C_{t'} \in \mathsf{C}\Lambda_{\oplus,\mathsf{let}}{}^{(\emptyset;\emptyset)}$ and $D_{t'} \in \mathsf{C}\Lambda_{\oplus,\mathsf{let}}{}^{(\emptyset;\emptyset)}$ such that for every $M \in \Lambda^{\emptyset}_{\oplus,\mathsf{let}}$ and $V \in \mathcal{V}^{\emptyset}_{\oplus,\mathsf{let}}$ we have that $P_{t'}(M) = \sum [\![C_{t'}[M]]\!]$ and $P_{t'}(\widetilde{V}) = \sum [\![D_{t'}[V]]\!]$. An action a can be either a closed term or a τ action, thus depending on it we differ two cases.

  1. If $a = \tau$, then a test t is of the form $\tau.t'$. From Definition 14 and Definition 22 we have $P_{\tau.t'}(\widetilde{V}) = 0$ for any value $V \in \mathcal{V}^{\emptyset}_{\oplus,\mathsf{let}}$. Hence, we define $D_{\tau.t'} = \Omega[\cdot]$ and the statement holds. Let M be a closed term. From the definition of a transition probability matrix ($P(M, \tau, \widetilde{V}) = [\![M]\!](V)$) and induction hypothesis $P_{t'}(\widetilde{V}) = \sum [\![D_{t'}[V]]\!]$ it follows that

  $$P_{\tau.t'}(M) = \sum_{\widetilde{V} \in \mathsf{V}\Lambda^{\emptyset}_{\oplus,\mathsf{let}}} P(M, \tau, \widetilde{V}) P_{t'}(\widetilde{V}) = \sum_{V \in \mathcal{V}^{\emptyset}_{\oplus,\mathsf{let}}} [\![M]\!](V) \cdot \sum [\![D_{t'}[V]]\!].$$

  We define $C_{\tau.t'} = (\mathsf{let}\ y = [\cdot]\ \mathsf{in}\ D_{t'}[y])$. Then, by the definition of operational semantics we get

  $$\sum [\![C_{\tau.t'}[M]]\!] = \sum [\![\mathsf{let}\ y = M\ \mathsf{in}\ D_{t'}[y]]\!] = \sum_{V \in \mathcal{V}^{\emptyset}_{\oplus,\mathsf{let}}} [\![M]\!](V) \cdot \sum [\![D_{t'}[V]]\!],$$

  for any closed term $M \in \Lambda^{\emptyset}_{\oplus,\mathsf{let}}$. Thus, $P_{\tau.t'}(M) = \sum [\![C_{\tau.t'}[M]]\!]$.

  2. If $a = F$ for some closed term F, then a test t is of the form $F.t'$. From Definition 14 and Definition 22 we have $P_{F.t'}(M) = 0$ for any term $M \in \Lambda^{\emptyset}_{\oplus,\mathsf{let}}$. Hence, we define $C_{F.t'} = \Omega[\cdot]$ and the statement holds. Let V be a value $\lambda x.N$ ($\widetilde{V} = \nu x.N$). From the definition of a transition probability matrix ($P(\nu x.N, F, N\{F/x\}) = 1$) and induction hypothesis, $P_{t'}(M) = \sum [\![C_{t'}[M]]\!]$ for every $M \in \Lambda^{\emptyset}_{\oplus,\mathsf{let}}$, it follows that

  $$\begin{aligned} P_{F.t'}(\widetilde{V}) &= \sum_{N' \in \Lambda^{\emptyset}_{\oplus,\mathsf{let}}} P(\widetilde{V}, F, N') P_{t'}(N') \\ &= P(\nu x.N, F, N\{F/x\}) \cdot P_{t'}(N\{F/x\}) \\ &= 1 \cdot P_{t'}(N\{F/x\}) = \sum [\![C_{t'}[N\{F/x\}]]\!] \end{aligned}$$

  By Lemma 6 terms $N\{F/x\}$ and $(\lambda x.N)F$ have the same semantics. Hence, they are bisimilar (Proposition 17). Due to the fact that bisimilarity is included in context equivalence (Theorem 20) we have that terms $N\{F/x\}$ and $(\lambda x.N)F$ are context equivalent. More precisely, for any context C, $\sum [\![C[N\{F/x\}]]\!] = \sum [\![C[(\lambda x.N)F]]\!]$. Finally, we obtain that

  $$P_{F.t'}(\widetilde{V}) = \sum [\![C_{t'}[N\{F/x\}]]\!] = \sum [\![C_{t'}[(\lambda x.N)F]]\!] = \sum [\![C_{t'}[VF]]\!].$$

  We define $D_{F.t'} = C_{t'}[[\cdot]F]$. Then, we have that $\sum [\![D_{F.t'}[V]]\!] = \sum [\![C_{t'}[VF]]\!]$, holds for any value $V \in \mathcal{V}^{\emptyset}_{\oplus,\mathsf{let}}$. Thus, $P_{F.t'}(\widetilde{V}) = \sum [\![D_{F.t'}[V]]\!]$.

- Finally, let $t = (t_1, t_2)$. By induction hypothesis there exist contexts $C_{t_1}, D_{t_1}, C_{t_2}, D_{t_2} \in \mathsf{C}\Lambda_{\oplus,\mathsf{let}}{}^{(\emptyset;\emptyset)}$ such that for any closed term M and a value V the following holds:

$$P_{t_1}(M) = \sum [\![C_{t_1}[M]]\!], \quad P_{t_1}(\widetilde{V}) = \sum [\![D_{t_1}[V]]\!],$$

$$P_{t_2}(M) = \sum [\![C_{t_2}[M]]\!] \quad \text{and} \quad P_{t_2}(\widetilde{V}) = \sum [\![D_{t_2}[V]]\!].$$

From Definition 22 we have

$$P_{(t_1,t_2)}(M) = P_{t_1}(M) \cdot P_{t_2}(M) = \sum [\![C_{t_1}[M]]\!] \cdot \sum [\![C_{t_2}[M]]\!],$$

for any closed term $M \in \Lambda^{\emptyset}_{\oplus,\mathsf{let}}$. We define:

$$C_{(t_1,t_2)} = (\lambda y.(\mathsf{let}\ z_1 = C_{t_1}[y]\ \mathsf{in}\ (\mathsf{let}\ z_2 = C_{t_2}[y]\ \mathsf{in}\ I)))[\cdot] \tag{8}$$

and by the definition of operational semantics we have

$$\sum [\![C_{(t_1,t_2)}[M]]\!] = \sum [\![C_{t_1}[M]]\!] \cdot \sum [\![C_{t_2}[M]]\!].$$

Since, for a value $V \in \mathcal{V}^{\emptyset}_{\oplus,\mathsf{let}}$ it holds that

$$P_{(t_1,t_2)}(\widetilde{V}) = P_{t_1}(\widetilde{V}) \cdot P_{t_2}(\widetilde{V}) = \sum [\![D_{t_1}[V]]\!] \cdot \sum [\![D_{t_2}[V]]\!],$$

we define $D_{(t_1,t_2)} = (\lambda y.(\mathsf{let}\ z_1 = D_{t_1}[y]\ \mathsf{in}\ (\mathsf{let}\ z_2 = D_{t_2}[y]\ \mathsf{in}\ I)))[\cdot]$ and the statement holds.

This concludes the proof. ◀

▶ **Lemma 26.** *Let $M, N \in \Lambda^{\emptyset}_{\oplus,\mathsf{let}}$, $M \leq N$ implies that $P_t(M) \leq P_t(N)$, for every test t.*

Proof. It is a straightforward consequence of Lemma 25. Let us assume that terms M and N are in the context preorder, $\emptyset \vdash M \leq N$. Then, for every context $C \in \mathsf{C}\Lambda_{\oplus,\mathsf{let}}{}^{(\emptyset;\emptyset)}$,we have $\sum [\![C[M]]\!] \leq \sum [\![C[N]]\!]$. By Lemma 25, we have that for each test $t \in \mathcal{T}_{\Lambda_{\oplus,\mathsf{let}}}$ there exists context C_t such that for every term M, $P_t(M) = \sum [\![C_t[M]]\!]$. Then, for every test $t \in \mathcal{T}_{\Lambda_{\oplus,\mathsf{let}}}$, it holds that $P_t(M) = \sum [\![C_t[M]]\!] \leq \sum [\![C_t[N]]\!] = P_t(N)$. Hence, for every test $t \in \mathcal{T}_{\Lambda_{\oplus,\mathsf{let}}}$ it holds that $P_t(M) \leq P_t(N)$. ◀

▶ **Theorem 27.** *Let $M, N \in \Lambda^{\emptyset}_{\oplus,\mathsf{let}}$, $M \simeq N$ implies that $P_t(M) = P_t(N)$, for every test t.*

Proof. It is a straightforward consequence of Lemma 26 and the fact that $M \simeq N$ is equivalent to $M \leq N$ and $N \leq M$. ◀

Notice that the let-in operator is crucial in defining the context $C_{(t_1,t_2)}$ associated with the product (t_1, t_2) of tests (Equation (8)) in the proof of Lemma 25. For example, if we consider the call-by-name version of $C_{(t_1,t_2)}$, i.e. the context $C = (\lambda y.(\lambda z_1 z_2.I) D_{t_1}[y] D_{t_2}[y])[\cdot]$, then the semantics of $C[M]$ is independent from the contexts $D_{t_1}[\cdot]$, $D_{t_2}[\cdot]$ and the term M, being $[\![C[M]]\!] = I$. Hence, we cannot have $P_{(t_1,t_2)}(M) = \sum [\![C[M]]\!]$ for every M. Another possibility is to try to use a context not erasing $D_{t_1}[\cdot]$ and $D_{t_2}[\cdot]$ during the evaluation, as for example in $C = (\lambda y.D_{t_1}[y] D_{t_2}[y])[\cdot]$. However this would imply to be able to control the result of $D_{t_1}[M]$ for every term M, for example supposing $[\![D_{t_1}[M]]\!] = P_{t_1}(M) I$, which increases considerably the difficulty of the proof. Anyway, the fact that there are examples of terms distinguished by tests (Example 23) but not by contexts without the let-in operator (Example 7) shows the necessity of this latter.

The following resumes all results in the paper, as sketched in Figure 1:

▶ **Corollary 28** (Full Abstraction). *For any $M, N \in \Lambda^{\emptyset}_{\oplus,\mathsf{let}}$, the following items are equivalent:*
(context equivalence) *$M \simeq N$,*
(bisimilarity) *$M \sim N$,*
(testing equivalence) *$P_t(M) = P_t(N)$ for all tests t.*

Concerning inequalities, the equivalence of similarity and testing preorder, i.e. a relation which contains (s, s') if and only if $P_t(s) \leq P_t(s')$ for every test $t \in \mathcal{T}_{(\mathcal{S},\mathcal{L},\mathcal{P})}$, does not hold as we stated below Theorem 24. So, we have no clue for proving that similarity is fully abstract with respect to the context preorder. A possible way to achieve this inequality full abstraction is to look for the converse of Lemma 25, by representing all contexts by tests. However, such a representation is far to be obvious, and even it might not exists.

5 Conclusion

In this paper we have considered the $\Lambda_{\oplus,\mathsf{let}}$-calculus, a pure untyped λ-calculus extended with two operators: a probabilistic choice operator $\oplus$ and a let-in operator. The calculus implements a lazy call-by-name evaluation strategy, following [1, 7], however the let-in operator allows for a call-by-value passing policy. We prove that context equivalence, bisimilarity and testing equivalence all coincide in $\Lambda_{\oplus,\mathsf{let}}$ (Corollary 28).

Concerning the inequalities associated with these equivalences: it is known that that the probabilistic similarity does not imply the testing approximation [16]. We prove that similarity implies context preorder (Theorem 20), but it remains open whether also the converse holds.

This paper confirms a conjecture stated in [3], showing that the calculus introduced in [7] can be endowed with a fully abstract bisimilarity by adding a let-in operator. As discussed in the Introduction, our feeling is that the need of this operator is due to the lazyness rather than to the cbn policy of the calculus. In order to precise this intuition we plan to investigate the definition of bisimilarity for the non-lazy cbn probabilistic λ-calculus, which has already fully abstract denotational models [2, 14] as well as infinitary normal forms [13] but not a theory of bisimulations.

In a more general perspective, one can study the let-in operator in call-by-name languages with different effects than the probabilistic one, as for example the non-determinism. Let us also mention [5], which considers a kind of applicative bisimulation for lazy call-by-name lambda-calculi endowed with various algebraic effects. However, the setting seems different, as in [5] the notion of (bi)simulation is not defined over terms but over their semantics.

Finally, one should address similar questions in typed languages. We have already mentioned in the Introduction that [10] shows that adding a let-in operator at ground types does not alter the observational equality of the cbn probabilistic PCF, but what about allowing a let-in at higher-order types? and what in presence of recursive types?

References

1 Samson Abramsky. *Research Topics in Functional Programming*, chapter The Lazy Lambda Calculus, pages 65–116. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990. URL: `http://dl.acm.org/citation.cfm?id=119830.119834`.

2 Pierre Clairambault and Hugo Paquet. Fully Abstract Models of the Probabilistic lambda-calculus. In Dan R. Ghica and Achim Jung, editors, *27th EACSL Annual Conference on Computer Science Logic, CSL 2018, September 4-7, 2018, Birmingham, UK*, volume 119 of *LIPIcs*, pages 16:1–16:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. `doi:10.4230/LIPIcs.CSL.2018.16`.

3. Raphaëlle Crubillé and Ugo Dal Lago. On Probabilistic Applicative Bisimulation and Call-by-Value λ-Calculi. In *Programming Languages and Systems - 23rd European Symposium on Programming, ESOP 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, pages 209–228, 2014. doi:10.1007/978-3-642-54833-8_12.
4. Raphaëlle Crubillé, Ugo Dal Lago, Davide Sangiorgi, and Valeria Vignudelli. On Applicative Similarity, Sequentiality, and Full Abstraction. In *Correct System Design - Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday, Oldenburg, Germany, September 8-9, 2015. Proceedings*, pages 65–82, 2015. doi:10.1007/978-3-319-23506-6_7.
5. Ugo Dal Lago, Francesco Gavazzo, and Ryo Tanaka. Effectful Applicative Similarity for Call-by-Name Lambda Calculi. In Dario Della Monica, Aniello Murano, Sasha Rubin, and Luigi Sauro, editors, *Joint Proceedings of the 18th Italian Conference on Theoretical Computer Science and the 32nd Italian Conference on Computational Logic co-located with the 2017 IEEE International Workshop on Measurements and Networking (2017 IEEE M&N), Naples, Italy, September 26-28, 2017.*, volume 1949 of *CEUR Workshop Proceedings*, pages 87–98. CEUR-WS.org, 2017.
6. Ugo Dal Lago, Davide Sangiorgi, and Michele Alberti. On Coinductive Equivalences for Higher-Order Probabilistic Functional Programs (Long Version). *CoRR*, abs/1311.1722, 2013. arXiv:1311.1722.
7. Ugo Dal Lago, Davide Sangiorgi, and Michele Alberti. On coinductive equivalences for higher-order probabilistic functional programs. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 297–308, 2014. doi:10.1145/2535838.2535872.
8. Ugo Dal Lago and Margherita Zorzi. Probabilistic operational semantics for the lambda calculus. *RAIRO - Theoretical Informatics and Applications - Informatique Théorique et Applications*, 46(3):413–450, 2012. doi:10.1051/ita/2012012.
9. Thomas Ehrhard, Michele Pagani, and Christine Tasson. The Computational Meaning of Probabilistic Coherence Spaces. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011, June 21-24, 2011, Toronto, Ontario, Canada*, pages 87–96, 2011. doi:10.1109/LICS.2011.29.
10. Thomas Ehrhard, Michele Pagani, and Christine Tasson. Full Abstraction for Probabilistic PCF. *Journal of the ACM*, 65(4):23:1–23:44, 2018. doi:10.1145/3164540.
11. Jean-Yves Girard. Linear Logic. *Theoretical Computer Science*, 50:1–102, 1987. doi:10.1016/0304-3975(87)90045-4.
12. Kim Guldstrand Larsen and Arne Skou. Bisimulation through Probabilistic Testing. *Information and Computation*, 94(1):1–28, 1991. doi:10.1016/0890-5401(91)90030-6.
13. Thomas Leventis. Probabilistic Böhm Trees and Probabilistic Separation. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 649–658, 2018. doi:10.1145/3209108.3209126.
14. Thomas Leventis and Michele Pagani. Strong Adequacy and Untyped Full Abstraction for Probabilistic Coherence Spaces. accepted to FOSSACS 2019, 2019.
15. Sungwoo Park, Frank Pfenning, and Sebastian Thrun. A Probabilistic Language Based on Sampling Functions. *ACM Transactions on Programming Languages and Systems*, 31(1):4:1–4:46, December 2008. doi:10.1145/1452044.1452048.
16. Franck Van Breugel, Michael W. Mislove, Joël Ouaknine, and James Worrell. Domain theory, testing and simulation for labelled Markov processes. *Theoretical Computer Science*, 333(1-2):171–197, 2005. doi:10.1016/j.tcs.2004.10.021.

A Appendix - Proofs

A.1 Context Equivalence is a Congruence

We consider $\Lambda_{\oplus,\mathsf{let}}$-relations defined in Section 3. The set $P_{\mathsf{FIN}}(X)$ denotes the set of all finite subsets of X.

▶ **Definition 29.** *A $\Lambda_{\oplus,\mathsf{let}}$-relation $\mathcal{R}$ is compatible if and only if the five conditions below hold:*

(Com1) $\forall \Gamma \in P_{\mathsf{FIN}}(X), x \in \Gamma : \Gamma \vdash x \mathrel{\mathcal{R}} x$;

(Com2) $\forall \Gamma \in P_{\mathsf{FIN}}(X), \forall x \in X - \Gamma,\ \forall M, N \in \Lambda^{\Gamma\cup\{x\}}_{\oplus,\mathsf{let}} : \Gamma \cup \{x\} \vdash M \mathrel{\mathcal{R}} N \Rightarrow \Gamma \vdash \lambda x.M \mathrel{\mathcal{R}} \lambda x.N$;

(Com3) $\forall \Gamma \in P_{\mathsf{FIN}}(X), \forall M, N, L, P \in \Lambda^{\Gamma}_{\oplus,\mathsf{let}} : \Gamma \vdash M \mathrel{\mathcal{R}} N \wedge \Gamma \vdash L \mathrel{\mathcal{R}} P \Rightarrow \Gamma \vdash ML \mathrel{\mathcal{R}} NP$;

(Com4) $\forall \Gamma \in P_{\mathsf{FIN}}(X), \forall M, N, L, P \in \Lambda^{\Gamma}_{\oplus,\mathsf{let}} : \Gamma \vdash M \mathrel{\mathcal{R}} N \wedge \Gamma \vdash L \mathrel{\mathcal{R}} P \Rightarrow \Gamma \vdash M \oplus L \mathrel{\mathcal{R}} N \oplus P$;

(Com5) $\forall \Gamma \in P_{\mathsf{FIN}}(X), \forall x \in X, \forall M, N \in \Lambda^{\Gamma}_{\oplus,\mathsf{let}}, \forall L, P \in \Lambda^{\Gamma\cup\{x\}}_{\oplus,\mathsf{let}} : \Gamma \vdash M \mathrel{\mathcal{R}} N \wedge \Gamma \cup \{x\} \vdash L \mathrel{\mathcal{R}} P \Rightarrow \Gamma \vdash (\mathsf{let}\ x = M\ \mathsf{in}\ L) \mathrel{\mathcal{R}} (\mathsf{let}\ x = N\ \mathsf{in}\ P)$.

▶ **Definition 30.** *A $\Lambda_{\oplus,\mathsf{let}}$-relation is a congruence (respectively, precongruence) if it is an equivalence relation (respectively, preorder) and compatible.*

This definition of a (pre)congruence is equivalent to Definition 18.

▶ **Lemma 31.** *The context preorder $\leq$ is a precongruence relation.*

Proof. The proof follows the same basic outline as the proof of Lemma 3.33 in [6]. ◀

▶ **Lemma 32.** *The context equivalence $\simeq$ is a congruence relation.*

Proof. This statement follows directly from Lemma 31 and the definition of context equivalence, i.e. $\simeq = \leq \cap (\leq)^{op}$. ◀

A.2 Bisimulation Equivalence is a Congruence

We use Howe's technique to prove that probabilistic similarity is a precongruence and as a consequence probabilistic bisimilarity is a congruence. Howe's technique is a commonly used technique for proving precongruence of similarity. The proof is very technical. It is the adaptation of the technique used in [3, 4, 6] and it has the same structure as the proof in [6]. Contrary to the proof in [6], our proof introduces a new notion of compatibility with the let-in operator.

The property $\sim = \lesssim \cap \lesssim^{op}$ ensures it is enough to show that probabilistic similarity ($\lesssim$) is a precongruence in order to prove that probabilistic bisimilarity ($\sim$) is a congruence. The key part is proving that $\lesssim$ is a compatible relation and it is done by Howe's technique.

We call an $\Lambda_{\oplus,\mathsf{let}}$-relation $\mathcal{R}$ *(term) substitutive* if for all $\Gamma \in P_{\mathsf{FIN}}(X), x \in X - \Gamma, M, N \in \Lambda^{\Gamma\cup\{x\}}_{\oplus,\mathsf{let}}, L, P \in \Lambda^{\Gamma}_{\oplus,\mathsf{let}}$ the following holds

$$\Gamma \cup \{x\} \vdash M \mathrel{\mathcal{R}} N \wedge \Gamma \vdash L \mathrel{\mathcal{R}} P \Rightarrow \Gamma \vdash M\{L/x\} \mathrel{\mathcal{R}} N\{P/x\}.$$

If a relation $\mathcal{R}$ satisfies

$$\Gamma \cup \{x\} \vdash M \mathrel{\mathcal{R}} N \wedge L \in \Lambda^{\Gamma}_{\oplus,\mathsf{let}} \Rightarrow \Gamma \vdash M\{L/x\} \mathrel{\mathcal{R}} N\{L/x\},$$

we say it is *closed under term-substitution.*

$$\frac{\Gamma \vdash x \ \mathcal{R} \ M}{\Gamma \vdash x \ \mathcal{R}^H \ M} \ (\text{How1}) \qquad \frac{\Gamma \cup \{x\} \vdash M \ \mathcal{R}^H \ L \quad \Gamma \vdash \lambda x.L \ \mathcal{R} \ N \quad x \notin \Gamma}{\Gamma \vdash \lambda x.M \ \mathcal{R}^H \ N} \ (\text{How2})$$

$$\frac{\Gamma \vdash M \ \mathcal{R}^H \ P \quad \Gamma \vdash N \ \mathcal{R}^H \ Q \quad \Gamma \vdash PQ \ \mathcal{R} \ L}{\Gamma \vdash MN \ \mathcal{R}^H \ L} \ (\text{How3})$$

$$\frac{\Gamma \vdash M \ \mathcal{R}^H \ P \quad \Gamma \vdash N \ \mathcal{R}^H \ Q \quad \Gamma \vdash P \oplus Q \ \mathcal{R} \ L}{\Gamma \vdash M \oplus N \ \mathcal{R}^H \ L} \ (\text{How4})$$

$$\frac{\Gamma \vdash M \ \mathcal{R}^H \ P \quad \Gamma \cup \{x\} \vdash N \ \mathcal{R}^H \ Q \quad \Gamma \vdash (\mathsf{let}\ x = P \ \mathsf{in}\ Q) \ \mathcal{R} \ L}{\Gamma \vdash (\mathsf{let}\ x = M \ \mathsf{in}\ N) \ \mathcal{R}^H \ L} \ (\text{How5})$$

Figure 5 Howe's lifting for $\Lambda_{\oplus,\mathsf{let}}$.

$$\frac{\Gamma \vdash M \ \mathcal{R} \ N}{\Gamma \vdash M \ \mathcal{R}^+ \ N} \ (\text{TC1})$$

$$\frac{\Gamma \vdash M \ \mathcal{R}^+ \ N \qquad \Gamma \vdash N \ \mathcal{R}^+ \ L}{\Gamma \vdash M \ \mathcal{R}^+ \ L} \ (\text{TC2})$$

Figure 6 Transitive closure for $\Lambda_{\oplus,\mathsf{let}}$.

Please notice that if $\mathcal{R}$ is substitutive and reflexive then it is closed under term-substitution. As stated in the paper, open extensions of $\lesssim$ and $\sim$ are closed under term-substitution by definition.

For an arbitrary $\Lambda_{\oplus,\mathsf{let}}$-relation $\mathcal{R}$, Howe's lifting $\mathcal{R}^H$ is defined by the rules in Figure 5. We start with some auxiliary statements.

▶ **Lemma 33.** *If $\mathcal{R}$ is reflexive, then $\mathcal{R}^H$ is compatible.*

Proof. The proof follows the same basic outline as the proof of Lemma 3.10 in [6]. ◀

▶ **Lemma 34.** *If $\mathcal{R}$ is transitive, then $\Gamma \vdash M \ \mathcal{R}^H \ N$ and $\Gamma \vdash N \ \mathcal{R} \ L$ imply $\Gamma \vdash M \ \mathcal{R}^H \ L$.*

Proof. By induction on the derivation of $\Gamma \vdash M \ \mathcal{R}^H \ N$, looking at the last rule used, thus on the structure of M. ◀

▶ **Lemma 35.** *If $\mathcal{R}$ is reflexive, then $\Gamma \vdash M \ \mathcal{R} \ N$ implies $\Gamma \vdash M \ \mathcal{R}^H \ N$.*

Proof. By induction on the structure of M. ◀

▶ **Lemma 36.** *If $\mathcal{R}$ is reflexive, transitive and closed under term-substitution, then $\mathcal{R}^H$ is (term) substitutive and hence also closed under term-substitution.*

Proof. We need to show that: $\forall \Gamma \in P_{\mathsf{FIN}}(X), \forall x \in X - \Gamma, \forall M, N \in \Lambda_{\oplus,\mathsf{let}}^{\Gamma \cup \{x\}}, \forall L, P \in \Lambda_{\oplus,\mathsf{let}}^{\Gamma}$,

$$\Gamma \cup \{x\} \vdash M \ \mathcal{R}^H \ N \ \wedge \ \Gamma \vdash L \ \mathcal{R}^H \ P \ \Rightarrow \Gamma \vdash M\{L/x\} \ \mathcal{R}^H \ N\{L/x\}.$$

Proof proceeds by induction on the derivation of $\Gamma \cup \{x\} \vdash M \ \mathcal{R}^H \ N$. ◀

The goal is to prove that $\lesssim^H$ is a precongruence, but in order to do that some properties are missing. Hence, following Howe's approach we build a transitive closure of a $\Lambda_{\oplus,\mathsf{let}}$-relation $\mathcal{R}$ as a relation $\mathcal{R}^+$ defined by the rules in Figure 6.

▶ **Lemma 37.** *If $\mathcal{R}$ is compatible, then so is $\mathcal{R}^+$.*

Proof. The proof follows the same basic outline as the proof of Lemma 3.14 in [6]. ◀

▶ **Lemma 38.** *If $\mathcal{R}$ is closed under term-substitution, then so is $\mathcal{R}^+$.*

Proof. Proving that $\mathcal{R}^+$ is closed under term-substitution means to show: $\forall \Gamma \in P_{\mathsf{FIN}}(X)$, $\forall x \in X - \Gamma$, $\forall M, N \in \Lambda_{\oplus,\mathsf{let}}^{\Gamma\cup\{\mathsf{x}\}}$, $\forall L \in \Lambda_{\oplus,\mathsf{let}}^{\Gamma}$, $\Gamma \cup \{x\} \vdash M \ \mathcal{R}^+ \ N \Rightarrow M\{L/x\} \ \mathcal{R}^+ \ N\{L/x\}$. Proof proceeds by induction on the derivation of $\Gamma \cup \{x\} \vdash M \ \mathcal{R}^+ \ N$. ◀

▶ **Lemma 39.** *If a $\Lambda_{\oplus,\mathsf{let}}$-relation $\mathcal{R}$ is a preorder, then so is $(\mathcal{R}^H)^+$.*

Proof. A relation is a preorder if it is reflexive and transitive. We assume that $\mathcal{R}$ is reflexive and transitive. Then, by Lemma 33 and Lemma 37 we conclude $(\mathcal{R}^H)^+$ is compatible and hence reflexive. Relation $(\mathcal{R}^H)^+$ is transitive by its construction, since it is a transitive closure of relation $\mathcal{R}^H$. Thus, we conclude relation $(\mathcal{R}^H)^+$ is a preorder. ◀

The crucial part in proving that probabilistic similarity is a precongruence is Key Lemma (Lemma 44). First, we need the definition of a probability assignment and an auxiliary lemma about it.

▶ **Definition 40.** $\mathbb{P} = (\{p_i\}_{1\leq i\leq n}, \{r_I\}_{I\subseteq\{1,\ldots,n\}})$ *is a probability assignment if for each* $I \subseteq \{1,\ldots,n\}$ *it holds that* $\sum_{i\in I} p_i \leq \sum_{J\cap I\neq\emptyset} r_J$.

▶ **Lemma 41.** *Let* $\mathbb{P} = (\{p_i\}_{1\leq i\leq n}, \{r_I\}_{I\subseteq\{1,\ldots,n\}})$ *be a probability assignment. Then for every nonempty* $I \subseteq \{1,\ldots,n\}$ *and for every* $k \in I$ *there is* $s_{k,I} \in [0,1]$ *which satisfies the following conditions:*

1. *for every I, it holds that* $\sum_{k\in I} s_{k,I} \leq 1$*;*
2. *for every* $k \in \{1,\ldots,n\}$*, it holds that* $p_k \leq \sum_{k\in I} s_{k,I} \cdot r_I$.

The proof of Lemma 41 is omitted, but it can be found in [6]. Besides Lemma 41, in the proof of Key Lemma we use the following technical Lemmas.

▶ **Lemma 42.** *For every* $X \subseteq \Lambda_{\oplus,\mathsf{let}}^{\{\mathsf{x}\}}$*, it holds that* $\precsim(\lambda x.X) = \lambda x.(\precsim(X))$ *and* $\precsim(\nu x.X) = \nu x.(\precsim(X))$. $\lambda x.(\precsim(X))$ *stands for the set* $\{\lambda x.M \mid \exists N \in X, N \precsim M\}$.

Proof. Straightforward consequence of the definition of similarity. ◀

▶ **Lemma 43.** *If* $M \precsim N$*, then for every* $X \subseteq \Lambda_{\oplus,\mathsf{let}}^{\{\mathsf{x}\}}$*,* $[\![M]\!](\lambda x.X) \leq [\![N]\!](\lambda x.\precsim(X))$.

Proof. It is a straightforward consequence of Lemma 42. ◀

▶ **Lemma 44** (Key Lemma)**.** *If* $M \precsim^H N$*, then for every* $X \subseteq \Lambda_{\oplus,\mathsf{let}}^{\{\mathsf{x}\}}$ *it holds that* $[\![M]\!](\lambda x.X) \leq [\![N]\!](\lambda x.(\precsim^H(X)))$.

Proof. Since $[\![M]\!] = \sup\{\mathcal{D} \ ; \ M \Downarrow \mathcal{D}\}$, it is enough to prove the following statement: if $M \precsim^H N$ and $M \Downarrow \mathcal{D}$ then for every $X \subseteq \Lambda_{\oplus,\mathsf{let}}^{\{\mathsf{x}\}}$ it holds that $\mathcal{D}(\lambda x.X) \leq [\![N]\!](\lambda x.(\precsim^H(X)))$. Proof proceeds by induction on the derivation of $M \Downarrow \mathcal{D}$, looking at the last rule used. We only consider the case where M is of the form $\mathsf{let}\ x = L\ \mathsf{in}\ P$, while the proofs of other cases follow the same basic outline as the proof of Lemma 3.17 in [6].

- Let $M = (\mathsf{let}\ x = L\ \mathsf{in}\ P)$. Then, we have $\mathcal{D} = \sum_{\lambda x.Q} \mathcal{F}(\lambda x.Q) \cdot \mathcal{H}_{Q,P}$ where $L \Downarrow \mathcal{F}$ and for any $\lambda x.Q \in \mathsf{S}(\mathcal{F})$, $\{P\{\lambda x.Q/x\} \Downarrow \mathcal{H}_{Q,P}\}$. The last rule used in the derivation of $\emptyset \vdash M \lesssim^H N$ has to be (How5), thus we get $\emptyset \vdash L \lesssim^H R$, $x \vdash P \lesssim^H S$ and $\emptyset \vdash (\mathsf{let}\ x = R\ \mathsf{in}\ S) \lesssim N$ as additional hypothesis. By applying the induction hypothesis on $L \Downarrow \mathcal{F}$ and $\emptyset \vdash L \lesssim^H R$, we obtain that

$$\mathcal{F}(\lambda x.Y) \leq [\![R]\!](\lambda x. \lesssim^H (Y)), \tag{9}$$

holds for any $Y \subseteq \Lambda^{\{x\}}_{\oplus,\mathsf{let}}$. $\mathcal{F}$ is a finite distribution, hence the distribution $\mathcal{D} = \sum_{\lambda x.Q} \mathcal{F}(\lambda x.Q) \cdot \mathcal{H}_{Q,P}$ is a sum of finitely many summands. Let the support of $\mathcal{F}$ be the set $\mathsf{S}(\mathcal{F}) = \{\lambda x.Q_1, \ldots, \lambda x.Q_n\}$. Equation (9) implies that for every $I \subseteq \{1, \ldots, n\}$ the following holds

$$\mathcal{F}(\bigcup_{i\in I} \lambda x.Q_i) \leq [\![R]\!](\bigcup_{i\in I} \lambda x. \lesssim^H (Q_i)).$$

This allows us to apply Lemma 41. Thus, for every $U \in \bigcup_{i=1}^n \lesssim^H (Q_i)$ there exist numbers $r_1^{U,R}, \ldots, r_n^{U,R}$ such that:

$$[\![R]\!](\lambda x.U) \geq \sum_{i=1}^n r_i^{U,R}, \qquad \forall U \in \bigcup_{i=1}^n \lesssim^H (Q_i);$$

$$\mathcal{F}(\lambda x.Q_i) \leq \sum_{U \in \lesssim^H (Q_i)} r_i^{U,R}, \qquad \forall i \in \{1, \ldots, n\}.$$

Now, we can conclude the following

$$\mathcal{D} \leq \sum_{i=1}^n \left(\sum_{U \in \lesssim^H (Q_i)} r_i^{U,R} \right) \cdot \mathcal{H}_{Q_i,P} = \sum_{i=1}^n \sum_{U \in \lesssim^H (Q_i)} r_i^{U,R} \cdot \mathcal{H}_{Q_i,P}.$$

Since $Q_i \lesssim^H U$ holds and $\lesssim^H$ is compatible by Lemma 33 , $\lambda x.Q_i \lesssim^H \lambda x.U$ holds. By applying Lemma 36 on $P \lesssim^H S$ and the latter we get $P\{\lambda x.Q_i/x\} \lesssim^H S\{\lambda x.U/x\}$. If we apply the induction hypothesis on the derivations $P\{\lambda x.Q_i/x\} \Downarrow \mathcal{H}_{Q_i,P}$, $i \in \{1, \ldots, n\}$, we obtain that for every $X \subseteq \Lambda^{\{x\}}_{\oplus,\mathsf{let}}$ it holds that

$$\mathcal{D}(\lambda x.X) \leq [\![N]\!](\lambda x. \lesssim^H (X)).$$

This concludes the proof. ◀

Proof of Lemma 19. The proof that similarity is a precongruence consists of two steps: the first step is to show that the relation $(\lesssim^H)^+$ is a precongruence and the second one is to show that it coincide with relation $\lesssim$. Since $\lesssim$ is a preorder, then by Lemma 39, relation $(\lesssim^H)^+$ is also a preorder. Relation $\lesssim$ is reflexive, hence by Lemma 33 we have $\lesssim^H$ is compatible. Furthermore, Lemma 37 ensures that $(\lesssim^H)^+$ is also compatible. So, we can conclude that $(\lesssim^H)^+$ is a precongruence. Next, we want to show that $\lesssim = (\lesssim^H)^+$. From the construction of Howe's lifting $\lesssim^H$ and its transitive closure $(\lesssim^H)^+$ it follows that $\lesssim \subseteq (\lesssim^H)^+$. It remains to show the inclusion $(\lesssim^H)^+ \subseteq \lesssim$. We show that $(\lesssim^H)^+$ is included in some probabilistic simulation $\mathcal{R}$, thus it is also included in the largest one, $\lesssim$. The relation we consider is $\mathcal{R} = \big\{(M, N) : M\ (\lesssim^H)^+\ N\} \cup \{(\nu x.M, \nu x.N) : M\ (\lesssim^H)^+\ N\big\}$. It is obvious that $(\lesssim^H)^+ \subseteq \mathcal{R}$, so it only remains to show that $\mathcal{R}$ is a probabilistic simulation. Relation $(\lesssim^H)^+$ is closed under term-substitution (by Lemma 36 and Lemma 38), hence it is enough to consider only closed terms and distinguished values. Since $(\lesssim^H)^+$ is a preorder relation (reflexive and transitive), it is easy to see $\mathcal{R}$ is also a preorder. We show the following two points:

1. If $M\ (\lesssim^H)^+\ N$, then for every $X \subseteq \Lambda^{\{x\}}_{\oplus,\mathsf{let}}$ it holds that $P(M,\tau,\nu x.X) \leq P(N,\tau,\mathcal{R}(\nu x.X))$.
2. If $M\ (\lesssim^H)^+\ N$, then for every $L \in \Lambda^{\emptyset}_{\oplus,\mathsf{let}}$ and for every $X \subseteq \Lambda^{\{x\}}_{\oplus,\mathsf{let}}$, $P(\nu x.M, L, X) \leq P(\nu x.N, L, \mathcal{R}(X))$.

The first point we prove by induction on the derivation of $M\ (\lesssim^H)^+\ N$. We look at the last rule used. Let us start with the base case where (TC1) is the last rule used. Then, we have $M \ \lesssim^H \ N$ holds by hypothesis. By Key Lemma we conclude the following:

$$\begin{aligned}P(M,\tau,\nu x.X) &= [\![M]\!](\lambda x.X)\\ &\leq [\![N]\!](\lambda x. \lesssim^H (X))\\ &\leq [\![N]\!](\lambda x.(\lesssim^H)^+(X))\\ &\leq [\![N]\!](\mathcal{R}(\nu x.X))\\ &= P(N,\tau,\mathcal{R}(\nu x.X)).\end{aligned}$$

Next, we consider the case where (TC2) is the last rule used and we have that for some $P \in \Lambda^{\emptyset}_{\oplus,\mathsf{let}}$, $M\ (\lesssim^H)^+\ P$ and $P\ (\lesssim^H)^+\ N$ hold. By induction hypothesis on both of them, we obtain:

$$P(M,\tau,X) \leq P(P,\tau,\mathcal{R}(X)),$$

$$P(P,\tau,\mathcal{R}(X)) \leq P(N,\tau,\mathcal{R}(\mathcal{R}(X))).$$

It is easy to show that $\mathcal{R}(\mathcal{R}(X)) \subseteq \mathcal{R}(X)$, thus we can conclude

$$P(M,\tau,X) \leq P(N,\tau,\mathcal{R}(X)).$$

This concludes the proof of the first point.

If $M\ (\lesssim^H)^+\ N$ and $L \in \Lambda^{\emptyset}_{\oplus,\mathsf{let}}$, then because of the fact that $(\lesssim^H)^+$ closed under term-substitution, we have that $M\{L/x\}\ (\lesssim^H)^+\ N\{L/x\}$ holds. As a consequence, we have that whenever $M\{L/x\} \in X$, then $N\{L/x\} \in (\lesssim^H)^+(X)$ and it holds that

$$\begin{aligned}P(\nu x.M, L, X) &= 1\\ &= P(\nu x.N, L, (\lesssim^H)^+(X))\\ &= P(\nu x.N, L, \mathcal{R}(X)).\end{aligned}$$

On the other hand, if $M\{L/x\} \notin X$, then $P(\nu x.M, L, X) = 0 \leq P(\nu x.N, L, \mathcal{R}(X))$.

To prove that bisimilarity is a congruence we need to prove that $\sim$ is an equivalence relation, which is compatible. Relation $\sim$ is an equivalence relation by its definition. Since we know that $\sim = \lesssim \cap \lesssim^{op}$ holds, from the fact that similarity is a precongruence it follows that $\sim$ is also compatible. This concludes the proof. ◀

Hilbert's Tenth Problem in Coq

Dominique Larchey-Wendling
Université de Lorraine, CNRS, LORIA, Vandœuvre-lès-Nancy, France
dominique.larchey-wendling@loria.fr

Yannick Forster
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
forster@ps.uni-saarland.de

Abstract

We formalise the undecidability of solvability of Diophantine equations, i.e. polynomial equations over natural numbers, in Coq's constructive type theory. To do so, we give the first full mechanisation of the Davis-Putnam-Robinson-Matiyasevich theorem, stating that every recursively enumerable problem – in our case by a Minsky machine – is Diophantine. We obtain an elegant and comprehensible proof by using a synthetic approach to computability and by introducing Conway's FRACTRAN language as intermediate layer.

2012 ACM Subject Classification Theory of computation → Models of computation; Theory of computation → Type theory

Keywords and phrases Hilbert's tenth problem, Diophantine equations, undecidability, computability theory, reduction, Minsky machines, Fractran, Coq, type theory

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.27

Supplement Material Coq formalisation of all results: `https://uds-psl.github.io/H10`, Coq library of undecidable problems: `https://github.com/uds-psl/coq-library-undecidability`

Funding *Dominique Larchey-Wendling*: partially supported by the TICAMORE project (ANR grant 16-CE91-0002).

Acknowledgements We would like to thank Gert Smolka, Dominik Kirst and Simon Spies for helpful discussion regarding the presentation.

1 Introduction

Hilbert's tenth problem (H10) was posed by David Hilbert in 1900 as part of his famous 23 problems [15] and asked for the "determination of the solvability of a Diophantine equation." A Diophantine equation[1] is a polynomial equation over natural numbers (or, equivalently, integers) with constant exponents, e.g. $x^2 + 3z = yz + 2$. When Hilbert asked for "determination," he meant, in modern terms, a decision procedure, but computability theory was yet several decades short of being developed.

The first undecidable problems found by Church, Post and Turing were either native to mathematical logic or dependent on a fixed model of computation. H10, to the contrary, can be stated to every mathematician and its formulation is independent from a model of computation. Emil Post stated in 1944 that H10 "begs for an unsolvability proof" [26]. From a computational perspective, it is clear that H10 is recursively enumerable (or *recognisable*), meaning there is an algorithm that halts on a Diophantine equation iff it is solvable.

[1] Named after the Greek mathematician Diophantus of Alexandria, who started the study of polynomial equations in the third century.

© Dominique Larchey-Wendling and Yannick Forster;
licensed under Creative Commons License CC-BY
4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019).
Editor: Herman Geuvers; Article No. 27; pp. 27:1–27:20
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Post's student Martin Davis conjectured that even the converse is true, i.e. that every recognisable set is also Diophantine. More precisely, he conjectured that if $A \subseteq \mathbb{N}^k$ is recognisable then $(a_1, \dots, a_k) \in A \leftrightarrow \exists x_1 \dots x_n, P(a_1, \dots, a_k, x_1, \dots, x_n) = 0$ holds for some polynomial P in $k+n$ variables. He soon improved on a result by Gödel [13] and gave a proof of his conjecture, however requiring up to one bounded universal quantification [3]: $(a_1, \dots, a_k) \in A \leftrightarrow \exists z, \forall y < z, \exists x_1 \dots x_n, P(a_1, \dots, a_k, x_1, \dots, x_n, y, z) = 0$. Davis and Putnam [5] further improved on this, and showed that, provided a certain number-theoretic assumption holds, every recognisable set is *exponentially* Diophantine, meaning variables are also allowed to appear in exponents. Julia Robinson then in 1961 modified the original proof to circumvent the need for the assumption, resulting in the DPR theorem [6], namely that every recognisable set is exponentially Diophantine. Due to another result from Robinson [27], the gap now only consisted of proving that there is a Diophantine equation exhibiting exponential growth. In 1970, Yuri Matiyasevich showed that the Fibonacci sequence grows exponentially while being Diophantine, closing the gap and finishing the proof of the theorem nowadays called *DPRM theorem*, ultimately establishing that exponentiation is Diophantine itself [19] (known as "Matiyasevich's theorem").

Even the most modern and simpler proofs of the DPRM theorem still require many preliminaries and complicated number-theoretic ideas, for an overview see [22]. We formalise one such proof as part of our ongoing work on a library of undecidable problems [10] in the proof assistant Coq [29]. Since H10 is widely used as a seed [7, 14] for showing the undecidability of problems using *many-one reductions*, this will open further ways of extending the library. Given that our library already contains a formalisation of Minsky machines [11], we follow the approach of Jones and Matijasevič [16], who use register machines, being very well-suited since they already work on numbers. They encode full computations of register machines as Diophantine equations in one single, monolithic step. To make the proof more tractable for both mechanisation and explanation, we factor out an intermediate language, John Conway's FRACTRAN [2], which can simulate Minsky machines.

We first introduce three characterisations of Diophantine equations over natural numbers, namely *Diophantine logic* DIO_FORM (allowing to connect Diophantine equations with conjunction, disjunction and existential quantification), *elementary Diophantine constraints* DIO_ELEM (a finite set of constraints on variables, oftentimes used for reductions [7, 14]) and *single Diophantine equations* DIO_SINGLE, including parameters, as described above. H10 then asks about the solvability of single Diophantine equations with no parameters.

Technically, the reduction chain to establish the unsolvability of H10 starts at the halting problem for single-tape Turing machines Halt, reduced to the Post correspondence problem PCP in [8]. In previous work [11] we have reduced PCP to a specialised halting problem for Minsky machines, which we use here in a slightly generalised form as MM. We then reduce Minsky machine halting to FRACTRAN termination. FRACTRAN is very natural to describe using polynomials, and the encoding does not rely on any complicated construction. The technical difficulty then only lies in the Diophantine encoding of the reflexive-transitive closure of a relation which follows from the direct elimination of bounded universal quantification, given that the proof in [20] involves no detour via models of computation. In total, we obtain the following chain of reductions to establish the undecidability of H10:

$$\text{Halt} \preceq \text{PCP} \preceq \text{MM} \preceq \text{FRACTRAN} \preceq \text{DIO_FORM} \preceq \text{DIO_ELEM} \preceq \text{DIO_SINGLE} \preceq \text{H10}$$

In the present paper, we focus on explaining this factorisation of the proof and give some details for the different stages. While we contribute Coq mechanisations of Matiyasevich's theorem and the elimination of bounded universal quantification, we treat them mainly as black-boxes and only elaborate on their challenging formalisation rather than the proofs themselves, a good explanation of which would anyways not fit in the given page limit.

To the best of our knowledge, we are the first to give a *full verification* of the DPRM theorem and the undecidability of Hilbert's tenth problem in a proof assistant. We base the notion of recognisability in the DPRM theorem on Minsky machines.

When giving undecidability proofs via many-one reductions, it is critical to show that all reduction functions are actually computable. We could in theory verify the computability of all functions involved using an explicit model of computation. In pen-and-paper proofs, this approach is however almost never used, because implementing high-level mathematical transformations as provably correct low-level programs is a daunting task. Instead, we rely on a synthetic approach [8, 9, 11] based on the computability of all functions definable in Coq's constructive type theory, which is closer to the practice of pen-and-paper proofs. In this approach, a problem P is considered undecidable if there is a reduction from an obviously undecidable problem, e.g. $\mathsf{Halt} \preceq P$.

The axiom-free Coq formalisation of all the results in this paper is available online and the main lemmas and theorems in the pdf version of the paper are hyper-linked with the html version of the source code at `https://uds-psl.github.io/H10`. Starting from our already existing library which included most of the Minsky machine code [11], the additional code for proving the undecidability of H10 and the DPRM theorem consists of about 8k loc including 3k loc for Matiyasevich's results alone, together with a 4k loc addition to our shared libraries; see Appendix A for more details. The paper itself can be read without in-depth knowledge of Coq or type theory.

Contribution. Apart from the full formalisation, we consider the novel refactoring of the proof via FRACTRAN a contribution to the explainability of the DPRM theorem.

Preliminaries. Regarding notation, we may write $x.y$ for multiplication of natural numbers $x, y : \mathbb{N}$ and we will leave out the symbol where convenient. We write $\mathbb{L}\,X$ for the type of *lists* over X and $l \mathbin{+\!\!+} l'$ for the *concatenation* of two lists. We write X^n for *vectors* $\vec{v}$ over type X with length n, and $\mathbb{F}_n$ for the *finite type* with exactly n elements. For $p : \mathbb{F}_n$, we write $\vec{v}[p]$ for the p-th component of $\vec{v} : X^n$. Notations for lists are overloaded for vectors. If $P : X \to \mathbb{P}$ is a predicate (on X) and $Q : Y \to \mathbb{P}$ is a predicate, we write $P \preceq Q$ if there is a function $f : X \to Y$ s.t. $\forall x : X,\ P\,x \leftrightarrow Q(f\,x)$, i.e. a *many-one reduction* from P to Q.

2 Diophantine Relations

Diophantine relations are composed of polynomials over natural numbers. There are several equivalent approaches to characterise these relations and oftentimes, the precise definition is omitted from papers. Basically, one can form equations between polynomial expressions and then combine these with conjunctions, disjunctions, and existential quantification.[2] For instance, these operations are assumed as Diophantine producing operators in e.g. [16, 19, 20, 21]. Sometimes, Diophantine relations are restricted to a single polynomial equation. Sometimes, the exponentiation function $x, y \mapsto x^y$ is assumed as Diophantine [16]. To complicate the picture, Diophantine relations might equivalently range over $\mathbb{Z}$ (instead of $\mathbb{N}$) but expressions like x^y implicitly assume that y never gets a negative value.

Although seemingly diverging, these approaches are not contradictory because in the end, they characterise the same class of relations on natural numbers. However, mechanisation does not allow for such implicit assumptions. To give some mechanisable structure to some

[2] Universal quantification or negation are not accepted as is.

of these approaches, we propose three increasingly restricted characterisations of Diophantine relations: *Diophantine logic*, *elementary Diophantine constraints* and *single Diophantine equations*, between which we provide computable transformations in Sections 3 and 4.

2.1 Diophantine Logic

We define the types $\mathbb{D}_{\text{expr}}$ of Diophantine expressions and $\mathbb{D}_{\text{form}}$ of Diophantine formulæ for the abstract syntax of Diophantine logic. Diophantine expressions are *polynomials* built from natural number constants and variables. An atomic Diophantine logic formula is just expressing the identity between two Diophantine expressions and we combine those with binary disjunction, binary conjunction, and existential quantification.

$$p, q : \mathbb{D}_{\text{expr}} ::= x_i : \mathsf{V} \mid n : \mathbb{N} \mid p \dot{+} q \mid p \dot{\times} q \qquad A, B : \mathbb{D}_{\text{form}} ::= p \doteq q \mid A \dot{\wedge} B \mid A \dot{\vee} B \mid \dot{\exists}\, A$$

The letters p, q ranges over expressions and the letters A, B range over formulæ. We use standard *De Bruijn syntax* with variables $x_0, x_1, \ldots$ of type $\mathsf{V} := \mathbb{N}$ for better readability. If we have $x_i : \mathsf{V}$, we write x_{1+i} for the next variable in V. As an example, the meta-level formula $\exists y, (y = 0 \wedge \exists z,\, y = z + 1)$ would be represented as $\dot{\exists}(x_0 \doteq 0 \mathbin{\dot{\wedge}} \dot{\exists}(x_1 \doteq x_0 \dot{+} 1))$, i.e. the variable x_i refers to the i-th binder in the context.

We provide a semantics for Diophantine logic. Given a valuation for variables $\nu : \mathsf{V} \to \mathbb{N}$, we define the interpretation $[\![p]\!]_\nu : \mathbb{N}$ of the expression $p : \mathbb{D}_{\text{expr}}$ by recursion:

$$[\![x_i]\!]_\nu := \nu\, x_i \qquad [\![n]\!]_\nu := n \qquad [\![p \dot{+} q]\!]_\nu := [\![p]\!]_\nu + [\![q]\!]_\nu \qquad [\![p \dot{\times} q]\!]_\nu := [\![p]\!]_\nu \times [\![q]\!]_\nu$$

The interpretation of formulæ cannot be done with a constant valuation $\nu : \mathsf{V} \to \mathbb{N}$ because of existential quantifiers. The interpretation $[\![A]\!]_\nu$ of the formula $A : \mathbb{D}_{\text{form}}$ is given by the following recursive rules:

$$\begin{array}{ll} [\![A \mathbin{\dot{\wedge}} B]\!]_\nu := [\![A]\!]_\nu \wedge [\![B]\!]_\nu & [\![p \doteq q]\!]_\nu := [\![p]\!]_\nu = [\![q]\!]_\nu \\ [\![A \mathbin{\dot{\vee}} B]\!]_\nu := [\![A]\!]_\nu \vee [\![B]\!]_\nu & [\![\dot{\exists}\, A]\!]_\nu := \exists n : \mathbb{N}, [\![A]\!]_{n\cdot\nu} \end{array} \qquad \text{with } \begin{cases} n\cdot\nu\,(x_0) := n \\ n\cdot\nu\,(x_{1+i}) := \nu\, x_i \end{cases}$$

where $n\cdot\nu : \mathsf{V} \to \mathbb{N}$ is the standard De Bruijn extension of a valuation ν by n.

We give a first formal characterisation of Diophantine polynomial expressions $\mathbb{D}_{\text{P}}$ and Diophantine relations $\mathbb{D}_{\text{R}}$. Diophantine polynomials are represented by some members of type $f : (\mathsf{V} \to \mathbb{N}) \to \mathbb{N}$ mapping valuations ν to values $f_\nu : \mathbb{N}$ which must moreover arise as instances of $\lambda\nu.[\![p]\!]_\nu$ for some $p : \mathbb{D}_{\text{expr}}$. And Diophantine relations are members of type $R : (\mathsf{V} \to \mathbb{N}) \to \mathbb{P}$ arising as instances of $\lambda\nu.[\![A]\!]_\nu$. We give an informative content to these sub-types of $(\mathsf{V} \to \mathbb{N}) \to \mathbb{N}$ and $(\mathsf{V} \to \mathbb{N}) \to \mathbb{P}$ to be able to do some computations with the witness (either p or A) of Diophantineness, typically when moving to another formal representation like elementary Diophantine constraints in Section 3.

▶ **Definition 1**. *We define the class of* Diophantine polynomials *and* Diophantine relations *as informative sub-types of* $(\mathsf{V} \to \mathbb{N}) \to \mathbb{N}$ *and* $(\mathsf{V} \to \mathbb{N}) \to \mathbb{P}$ *respectively:*

$$\mathbb{D}_{\text{P}}\, f := \textstyle\sum p : \mathbb{D}_{\text{expr}}, \big(\forall \nu, [\![p]\!]_\nu = f_\nu\big) \qquad \mathbb{D}_{\text{R}}\, R := \textstyle\sum A : \mathbb{D}_{\text{form}}, \big(\forall \nu, [\![A]\!]_\nu \leftrightarrow R\ \nu\big)$$

Note that Σ denotes type-theoretic dependent pairs. Hence an inhabitant w of $\mathbb{D}_{\text{R}}\, R$ is a (dependent) pair (A, H_A) where $A = \pi_1(w)$ is a Diophantine formula and $H_A = \pi_2(w)$ a proof that $[\![A]\!]_{(\cdot)}$ and R are extensionally equivalent. With these definitions, we will show that the sub-types $\mathbb{D}_{\text{P}}$ and $\mathbb{D}_{\text{R}}$ have the desired closure properties: $\mathbb{D}_{\text{P}}$ contains variables, constants and is closed under the $+$ and $\times$ pointwise operators over $(\mathsf{V} \to \mathbb{N}) \to \mathbb{N}$; $\mathbb{D}_{\text{R}}$ contains polynomial equations and is closed under conjunction, disjunction and existential quantification.

▶ **Proposition 2.** *Let* $x_i : \mathsf{V}$*,* $n : \mathbb{N}$*, and* $f, g : (\mathsf{V} \to \mathbb{N}) \to \mathbb{N}$ *be s.t.* $\mathbb{D}_\mathrm{P}\, f$ *and* $\mathbb{D}_\mathrm{P}\, g$ *hold. Then* $\mathbb{D}_\mathrm{P}\,(\lambda\nu.\nu\, x_i)$*,* $\mathbb{D}_\mathrm{P}\,(\lambda\nu.n)$*,* $\mathbb{D}_\mathrm{P}\,(\lambda\nu.f_\nu + g_\nu)$ *and* $\mathbb{D}_\mathrm{P}\,(\lambda\nu.f_\nu \times g_\nu)$ *hold.*

▶ **Proposition 3.** *Let* f*,* g *be s.t.* $\mathbb{D}_\mathrm{P}\, f$ *and* $\mathbb{D}_\mathrm{P}\, g$ *hold. Then* $\mathbb{D}_\mathrm{R}\,(\lambda\nu.\mathtt{True})$*,* $\mathbb{D}_\mathrm{R}\,(\lambda\nu.\mathtt{False})$*,* $\mathbb{D}_\mathrm{R}\,(\lambda\nu.f_\nu = g_\nu)$*,* $\mathbb{D}_\mathrm{R}\,(\lambda\nu.f_\nu \le g_\nu)$*,* $\mathbb{D}_\mathrm{R}\,(\lambda\nu.f_\nu < g_\nu)$ *and* $\mathbb{D}_\mathrm{R}\,(\lambda\nu.f_\nu \ne g_\nu)$ *hold.*

Proof. For e.g. $\lambda\nu.\, f_\nu < g_\nu$, we first get the witnesses for $w_f : \mathbb{D}_\mathrm{P}\, f$ and $w_g : \mathbb{D}_\mathrm{P}\, g$ by the projections $p_f := \pi_1(w_f)$ and $p_g := \pi_1(w_g)$. If we denote by ρ the "lift by one renaming" $\rho := \lambda x_i.x_{1+i}$ and then the witness $\dot\exists\big(1 \dot+ x_0 \dot+ \rho\,(p_f) \doteq \rho\,(p_g)\big)$ can be used for $\lambda\nu.\, f_\nu < g_\nu$. ◀

From a *mechanisation point of view,* having to provide explicit witnesses is a painful task and we now describe how it can be almost entirely automated. We use the Coq unification mechanism to analyse a meta-level expression of *Diophantine shape* and reflect it into the corresponding object-level witness of types either $\mathbb{D}_\mathrm{expr}$ or $\mathbb{D}_\mathrm{form}$ together with the proof that it is an appropriate witness. The following lemma provides a way to process a goal such as $\mathbb{D}_\mathrm{R}\, R$ depending on the meta-level syntax of R.

▶ **Lemma 4.** *Let* $R, S : (\mathsf{V} \to \mathbb{N}) \to \mathbb{P}$ *and* $T : \mathbb{N} \to (\mathsf{V} \to \mathbb{N}) \to \mathbb{P}$*. We have the maps:*
1. $\mathbb{D}_\mathrm{R}\, R \to \mathbb{D}_\mathrm{R}\, S \to \mathbb{D}_\mathrm{R}(\lambda\nu.R\,\nu \wedge S\,\nu)$
2. $\mathbb{D}_\mathrm{R}\, R \to \mathbb{D}_\mathrm{R}\, S \to \mathbb{D}_\mathrm{R}(\lambda\nu.R\,\nu \vee S\,\nu)$
3. $\mathbb{D}_\mathrm{R}\big(\lambda\nu.T\,(\nu\, x_0)\,(\lambda x_i.\nu\, x_{1+i})\big) \to \mathbb{D}_\mathrm{R}\big(\lambda\nu.\exists u, T\, u\,\nu\big)$
4. $(\forall\nu, S\,\nu \leftrightarrow R\,\nu) \to \mathbb{D}_\mathrm{R}\, R \to \mathbb{D}_\mathrm{R}\, S$.

With maps 1–3, we cope with conjunction, disjunction, existential quantification. Atomic or already established Diophantine relations are captured by Propositions 2 and 3 or later established results which are declared as *hints* for Coq proof-search tactics. The map number 4 provides a way to replace $\mathbb{D}_\mathrm{R}\, S$ with $\mathbb{D}_\mathrm{R}\, R$ once a proof that they are logically equivalent is established. Hence, if S cannot be analysed because it does not currently have a Diophantine shape, it can still be replaced by an equivalent relation R, hopefully better behaved.

2.2 Example of a Mechanised Diophantineness Proof

With the example of the "does not divide" relation $u \nmid v := \neg(\exists k, v = k \times u)$, we describe how to use those results to automate the production of the object-level $\mathbb{D}_\mathrm{R}$ witness A of Definition 1 from the meta-level representation of a relation of Diophantine shape.

▶ **Proposition 5.** $\forall f\, g : (\mathsf{V} \to \mathbb{N}) \to \mathbb{N},\ \mathbb{D}_\mathrm{P}\, f \to \mathbb{D}_\mathrm{P}\, g \to \mathbb{D}_\mathrm{R}\,(\lambda\nu.f_\nu \nmid g_\nu)$.

Proof. $u \nmid v := \neg(\exists k, v = k \times u)$ obviously is not in Diophantine shape. We thus first prove the equivalence $u \nmid v \leftrightarrow (u = 0 \wedge v \ne 0 \vee \exists a\, b,\ v = a \times u + b \wedge 0 < b < u)$ and this new expression now has a Diophantine shape, relying on the Diophantine shape of Euclidean division. Using this equivalence in combination with map 4 of Lemma 4, we replace the goal $\mathbb{D}_\mathrm{R}\,(\lambda\nu.f_\nu \nmid g_\nu)$ with $\mathbb{D}_\mathrm{R}\,(\lambda\nu.f_\nu = 0 \wedge g_\nu \ne 0 \vee \exists a\, b,\ g_\nu = a \times f_\nu + b \wedge 0 < b \wedge b < f_\nu)$ and then apply maps 1–3 of Lemma 4 until a shape such as those of Proposition 3 appears. ◀

Once established, we can add the map $\mathbb{D}_\mathrm{P}\, f \to \mathbb{D}_\mathrm{P}\, g \to \mathbb{D}_\mathrm{R}\,(\lambda\nu.f_\nu \nmid g_\nu)$ in the Diophantine hint database so that later encountered proof goals $\mathbb{D}_\mathrm{R}\,(\lambda\nu.f_\nu \nmid g_\nu)$ can be immediately solved. We implemented the Coq tactic `dio_rel_auto` to automate all this work. Apart from the equivalence for $u \nmid v$ and its proof, which cannot be guessed, the rest is effortless.

The recovery of witnesses of Definition 1 from meta-level syntax is automatic and hidden by the use of the `dio_rel_auto` tactic associated with the ever growing hint database. This

way, we can proceed as in e.g. Matiyasevich papers where he just transforms a relation into an equivalent Diophantine shape, accumulating more and more Diophantine shapes on the way. This is a very welcome simplification over having to program witnesses by hand.

2.3 Exponentiation and Bounded Universal Quantification

For now, we introduce the *elimination of the exponential relation* and then of *bounded universal quantification* as black boxes expressed in the theory of Diophantine relations. However we do contribute implementations for both of these hard results. It is not possible for these two mechanised proofs to be described in detail given the page limit. Nonetheless we postpone some remarks and discussions about these proofs in Section 5.

▶ **Theorem 6** (Exponential). $\forall f\,g\,h,\, \mathbb{D}_{\mathrm{P}}\, f \to \mathbb{D}_{\mathrm{P}}\, g \to \mathbb{D}_{\mathrm{P}}\, h \to \mathbb{D}_{\mathrm{R}}\big(\lambda\nu.f_\nu = g_\nu^{h_\nu}\big)$.

To prove it, one needs a meta-level Diophantine shape for the exponential relation, *the proof of which is nothing short of extraordinary.* This landmark result is due to Matiyasevich [19], but we have implemented the shorter and more up-to-date proof of [21].

▶ **Theorem 7** (Bounded U. Quantification). *For* $f : (\mathsf{V} \to \mathbb{N}) \to \mathbb{N}$ *and* $T : \mathbb{N} \to (\mathsf{V} \to \mathbb{N}) \to \mathbb{P}$*, we have a map* $\mathbb{D}_{\mathrm{P}}\, f \to \mathbb{D}_{\mathrm{R}}\big(\lambda\nu.T\,(\nu\, x_0)\ (\lambda x_i.\nu\, x_{1+i})\big) \to \mathbb{D}_{\mathrm{R}}\big(\lambda\nu.\forall u, u < f_\nu \to T\, u\, \nu\big)$.

This map can be compared with map 3 of Lemma 4 and allows to recognise bounded universal quantification as a legitimate Diophantine shape. We have implemented the direct proof of Matiyasevich [20] which does not involve a detour through a model of computation. Notice that the bound f_ν in $\forall u,\, u < f_\nu \to \ldots$ is not constant otherwise the elimination of the quantifier would proceed as a simple reduction to a finitary conjunction.

2.4 Reflexive-Transitive Closure is Diophantine

With these tools – elimination of the exponential relation and of bounded universal quantification – we can show that the reflexive and transitive closure of a Diophantine binary relation is itself Diophantine. We assume a binary relation $R : \mathbb{N} \to \mathbb{N} \to \mathbb{P}$ over natural numbers. The Diophantineness of R can be formalised by assuming that e.g. $\lambda\nu.R\ (\nu\, x_1)\ (\nu\, x_0)$ is a Diophantine relation. We show that the i-th iterate of R is Diophantine (where i is non-constant).

▶ **Lemma 8.** *Under hypothesis* $H_R : \mathbb{D}_{\mathrm{R}}\big(\lambda\nu.R\ (\nu\, x_1)\ (\nu\, x_0)\big)$*, for any* $f, g, i : (\mathsf{V} \to \mathbb{N}) \to \mathbb{N}$ *we have a map* $\mathbb{D}_{\mathrm{P}}\, f \to \mathbb{D}_{\mathrm{P}}\, g \to \mathbb{D}_{\mathrm{P}}\, i \to \mathbb{D}_{\mathrm{R}}\big(\lambda\nu.R^{i_\nu}\ f_\nu\ g_\nu\big)$.

Proof. Using Euclidean division, we define the `is_digit` $c\ q\ n\ d$ predicate stating that d is the n-th digit of the base q development of number c, as a Diophantine sentence:

$$\texttt{is_digit}\ c\ q\ n\ d := d < q \land \exists a\, b\, t,\ t = q^n \land c = (a.q + d).t + b \land b < t$$

The Diophantineness of this follows from Theorem 6. Then we define the `is_seq` $R\ c\ q\ i$ predicate stating that the first $i+1$ digits of c in base q form an R-chain, again with a Diophantine expression by H_R and Theorem 7:

$$\texttt{is_seq}\ R\ c\ q\ i := \forall n,\ n < i \to \exists u\, v,\ \texttt{is_digit}\ c\ q\ n\ u \land \texttt{is_digit}\ c\ q\ (1+n)\ v \land R\ u\ v$$

Then we encode $R^i\ u\ v$ by stating that there exists a (large enough) q and a number c such that the first $i+1$ digits of c in base q form an R-chain starting at u and ending at v:

$$R^i\ u\ v\ \leftrightarrow\ \exists q\, c,\ \texttt{is_seq}\ R\ c\ q\ i \land \texttt{is_digit}\ c\ q\ 0\ u \land \texttt{is_digit}\ c\ q\ i\ v$$

and this expression is accepted as Diophantine by Lemma 4. ◀

We fill in Lemma 8 in the Diophantine hint database and we derive the Diophantineness of the reflexive-transitive closure as a direct consequence of the equivalence $R^* \; u \; v \leftrightarrow \exists i, R^i \; u \; v$.

▶ **Theorem 9.** *For any binary relation $R : \mathbb{N} \to \mathbb{N} \to \mathbb{P}$ and any $f, g : (\mathsf{V} \to \mathbb{N}) \to \mathbb{N}$, we have the map $\mathbb{D}_{\mathrm{P}} \, f \to \mathbb{D}_{\mathrm{P}} \, g \to \mathbb{D}_{\mathrm{R}}\big(\lambda\nu.R \; (\nu \, x_1) \; (\nu \, x_0)\big) \to \mathbb{D}_{\mathrm{R}}\big(\lambda\nu.R^* \; f_\nu \; g_\nu\big)$.*

3 Elementary Diophantine Constraints

Elementary Diophantine constraints are very simple equations where only one instance of either $\dot{+}$ or $\dot{\times}$ is allowed. We give a direct proof that any Diophantine logic formula is semantically equivalent to the satisfiability of a list of elementary Diophantine constraints.

Starting from two copies of $\mathbb{N}$, one called U with u, v, w ranging over U for existentially quantified variables, and another one $\mathsf{V} = \{x_0, x_1, \ldots\}$ for parameters, we define the type of elementary Diophantine constraints by:

$$c : \mathbb{D}_{\mathrm{cstr}} ::= u \doteq n \mid u \doteq v \mid u \doteq x_i \mid u \doteq v \,\dot{+}\, w \mid u \doteq v \,\dot{\times}\, w \qquad \text{where } n : \mathbb{N}$$

Notice that these constraints do not have a "real" inductive structure, they are flat and of size either 3 or 5. Given two interpretations, $\varphi : \mathsf{U} \to \mathbb{N}$ for variables and $\nu : \mathsf{V} \to \mathbb{N}$ for parameters, it is trivial to define the semantics $[\![c]\!]^\varphi_\nu : \mathbb{P}$ of a single constraint c of type $\mathbb{D}_{\mathrm{cstr}}$:

$$[\![u \doteq n]\!]^\varphi_\nu := \varphi\, u = n \qquad [\![u \doteq v]\!]^\varphi_\nu := \varphi\, u = \varphi\, v \qquad [\![u \doteq v \,\dot{+}\, w]\!]^\varphi_\nu := \varphi\, u = \varphi\, v + \varphi\, w$$
$$[\![u \doteq x_i]\!]^\varphi_\nu := \varphi\, u = \nu\, x_i \qquad [\![u \doteq v \,\dot{\times}\, w]\!]^\varphi_\nu := \varphi\, u = \varphi\, v \times \varphi\, w$$

Given a list $l : \mathbb{L}\,\mathbb{D}_{\mathrm{cstr}}$ of constraints, we write $[\![l]\!]^\varphi_\nu$ when all the constraints in l are simultaneously satisfied, i.e. $[\![l]\!]^\varphi_\nu := \forall c, \, c \in l \to [\![c]\!]^\varphi_\nu$. We show the following result:

▶ **Theorem 10.** *For any Diophantine formula $A : \mathbb{D}_{\mathrm{form}}$ one can compute a list of elementary Diophantine constraints $l : \mathbb{L}\,\mathbb{D}_{\mathrm{cstr}}$ such that $\forall \nu : \mathsf{V} \to \mathbb{N}, \, [\![A]\!]_\nu \leftrightarrow \exists \varphi : \mathsf{U} \to \mathbb{N}, \, [\![l]\!]^\varphi_\nu$.*

I.e. for any given interpretation of parameters ν, $[\![A]\!]_\nu$ holds if and only if the constraints in l are simultaneously satisfiable. Hence any Diophantine logic formula is equivalent to the satisfiability of the conjunction of finitely many elementary Diophantine constraints.

The proof of Theorem 10 spans the rest of this section. We will strengthen the result a bit to be able to get an easy argument by induction on A.

▶ **Definition 11.** *Given a relation $R : (\mathsf{V} \to \mathbb{N}) \to \mathbb{P}$ and an interval $[u_a, u_{a+n}[\; \subseteq \mathsf{U}$, a* representation of R in *$[u_a, u_{a+n}[$ is given by:*

1. *a list $\mathcal{E} : \mathbb{L}\,\mathbb{D}_{\mathrm{cstr}}$ of constraints and a reference variable $\mathfrak{r} : \mathsf{U}$;*
2. *proofs that $\mathfrak{r}$ and the (existentially quantified) variables occurring in $\mathcal{E}$ belong to $[u_a, u_{a+n}[$;*
3. *a proof that the constraints in $\mathcal{E}$ are always (simultaneously) satisfiable, i.e. $\forall\nu\exists\varphi \, [\![\mathcal{E}]\!]^\varphi_\nu$;*
4. *a proof that the list $(\mathfrak{r} \doteq 0) :: \mathcal{E}$ is equivalent to R, i.e. $\forall \nu, \, R \; \nu \leftrightarrow \big(\exists \varphi, \, \varphi\,\mathfrak{r} = 0 \wedge [\![\mathcal{E}]\!]^\varphi_\nu\big)$.*

It is obvious that a representation of $\lambda\nu.[\![A]\!]_\nu$ in any interval $[u_a, u_{a+n}[$ is enough to prove Theorem 10 because of item 4 of Definition 11. But actually, computing such a representation is simpler than proving Theorem 10 directly.[3]

▶ **Lemma 12.** *For any $a : \mathbb{N}$ and any $A : \mathbb{D}_{\mathrm{form}}$, one can compute a representation of the relation $\lambda\nu.[\![A]\!]_\nu$ in $[u_a, u_{a+n}[$ for some value $n \leq 4|A|$.*[4]

---

[3] Proving Theorem 10 directly involves renamings of existential variables and might produce exponential blow-up in the number of constraints when handled naively.

[4] We denote the size of A with $|A|$. The actual statement in the code is a bit more complicated because we also show that the number of elementary constraints can be bounded by $1 + 3|A|$.

Proof. We show the result by structural induction on A.

- If A is $p \doteq q$ with $p, q : \mathbb{D}_{\text{expr}}$ then we encode p and q as a directed list of constraints. See Appendix B for a detailed explanation on an example;
- When A is $B \dot\wedge C$, we get a representation in $[u_a, u_{a+n_A}[$ by induction. Hence, let $(\mathcal{E}_B, \mathfrak{r}_B)$ be the representation of B in $[u_a, u_{a+n_B}[$. Then, inductively again, let $(\mathcal{E}_C, \mathfrak{r}_C)$ be a representation of C at $[u_{a+n_B}, u_{a+n_B+n_C}[$. We define $\mathfrak{r}_A := u_{a+n_A+n_B}$ and $\mathcal{E}_A := (\mathfrak{r}_A \doteq \mathfrak{r}_B \dot+ \mathfrak{r}_C) :: \mathcal{E}_B +\!\!+ \mathcal{E}_C$ and then $(\mathcal{E}_A, \mathfrak{r}_A)$ represents $A = B \dot\wedge C$ in $[u_a, u_{a+1+n_B+n_C}[$;[5]
- The case of $B \dot\vee C$ is similar: simply replace $\mathfrak{r}_A \doteq \mathfrak{r}_B \dot+ \mathfrak{r}_C$ with $\mathfrak{r}_A \doteq \mathfrak{r}_B \dot\times \mathfrak{r}_C$;
- We finish with the case when A is $\dot\exists B$. Let $(\mathcal{E}_B, \mathfrak{r}_B)$ be a representation of B in $[u_a, u_{a+n_B}[$. Let σ be the substitution mapping parameters in V and defined by $\sigma(x_0) := u_{a+n_B}$ and $\sigma(x_{1+i}) := x_i$; existential variables in U are left unmodified by this substitution. Then $(\sigma(\mathcal{E}_B), \mathfrak{r}_B)$ is a representation of $A = \dot\exists B$ in $[u_a, u_{a+1+n_B}[$.

This concludes the recursive construction of a representation of $\lambda\nu.[\![A]\!]_\nu$. ◀

4 Single Diophantine Equations

In this section, we show how a list of elementary Diophantine constraints can be simulated by a single identity between two Diophantine polynomials. We use the following well known convexity identity to achieve the reduction, the proof of which can be found in Appendix C.

▶ **Proposition 13.** *Let $(p_1, q_1), \dots, (p_n, q_n)$ be a sequence of pairs in $\mathbb{N} \times \mathbb{N}$. Then*

$$\sum_{i=1}^{n} 2p_i q_i = \sum_{i=1}^{n} p_i^2 + q_i^2 \ \leftrightarrow\ p_1 = q_1 \wedge \dots \wedge p_n = q_n.$$

We define Diophantine polynomials similar to the Diophantine expressions $\mathbb{D}_{\text{expr}}$ of Section 2.1 except that we now distinguish the types of bound variables (i.e. U) and of parameters (or free variables) (i.e. V) and that the types U and V are not fixed copies of $\mathbb{N}$ anymore, but type parameters of arbitrary value.

▶ **Definition 14.** *The type of* Diophantine polynomials $\mathbb{D}_{\text{poly}}(\mathsf{U}, \mathsf{V})$ *and the type of* single Diophantine equations $\mathbb{D}_{\text{single}}(\mathsf{U}, \mathsf{V})$ *are defined by:*

$$p, q : \mathbb{D}_{\text{poly}}(\mathsf{U}, \mathsf{V}) ::= u : \mathsf{U} \mid x_i : \mathsf{V} \mid n : \mathbb{N} \mid p \dot+ q \mid p \dot\times q \qquad E : \mathbb{D}_{\text{single}}(\mathsf{U}, \mathsf{V}) ::= p \doteq q$$

For $\varphi : \mathsf{U} \to \mathbb{N}$ and $\nu : \mathsf{V} \to \mathbb{N}$ we define the semantic interpretations of polynomials $[\![p]\!]_\nu^\varphi : \mathbb{N}$ and single Diophantine equations $[\![E]\!]_\nu^\varphi : \mathbb{P}$ in the obvious way.

▶ **Theorem 15.** *For any list $l : \mathbb{L}\,\mathbb{D}_{\text{cstr}}$ of elementary Diophantine constraints, one can compute a single Diophantine equation $E : \mathbb{D}_{\text{single}}(\mathbb{N}, \mathbb{N})$ such that $\forall \nu \forall \varphi,\ [\![E]\!]_\nu^\varphi \leftrightarrow [\![l]\!]_\nu^\varphi$.*

Proof. We write $l = [p_1 \doteq q_1; \dots; p_n \doteq q_n]$ and then use Proposition 13. In the code, we moreover show that the size of E is linear in the length of l. If needed, one could also show that the degree of the polynomial is less than 4. ◀

▶ **Corollary 16.** *Let $R : (\mathsf{V} \to \mathbb{N}) \to \mathbb{P}$. Assuming $\mathbb{D}_{\mathrm{R}}\, R$, one can compute a single Diophantine equation $p \doteq q : \mathbb{D}_{\text{single}}(\mathbb{N}, \mathsf{V})$ such that $\forall \nu,\ R\ \nu \leftrightarrow \exists \varphi,\ [\![p]\!]_\nu^\varphi = [\![q]\!]_\nu^\varphi$.*

---

[5] Since the intervals $[u_a, u_{a+n_B}[$ and $[u_{a+n_B}, u_{a+n_B+n_C}[$ are built disjoint, there is no difficulty in merging valuations whereas this usually involves renamings when existential variables are not carefully chosen.

Proof. Direct combination of Definition 1 and Theorems 10 and 15. In the formalisation, we also show that the size of the obtained single Diophantine equation is linearly bounded by the size of the witness formula contained in the proof of $\mathbb{D}_{\mathrm{R}}\, R$. ◀

We have shown that the automation we designed to recognise relations of Diophantine shape entail that these relations are also definable by satisfiability of a single equation between Diophantine polynomials, so these tools are sound w.r.t. a formally restrictive characterisation of Diophantineness. One could argue that the above existential quantifier $\exists\varphi$ encodes infinitely many existential quantifiers but it can easily be replaced by finitely many existential quantifiers over the bound variables that actually occur in p or q.

▶ **Proposition 17.** *For any single Diophantine equation $p \doteq q : \mathbb{D}_{\mathrm{single}}(\mathbb{N}, \mathsf{V})$, one can compute $n : \mathbb{N}$ and a new single Diophantine equation $p' \doteq q' : \mathbb{D}_{\mathrm{single}}(\mathbb{F}_n, \mathsf{V})$ such that for any $\nu : \mathsf{V} \to \mathbb{N}$, $(\exists \varphi : \mathbb{N} \to \mathbb{N}, [\![p]\!]^\varphi_\nu = [\![q]\!]^\varphi_\nu) \leftrightarrow (\exists \varphi : \mathbb{F}_n \to \mathbb{N}, [\![p']\!]^\varphi_\nu = [\![q']\!]^\varphi_\nu)$.*

Proof. We choose n greater that the number of bound variables which occur in either p or q. Then this subset of $\mathbb{N}$ can be faithfully embedded into the finite type $\mathbb{F}_n$ and we use such a renaming to compute (p', q'). Remark that the size of (p', q') is the same as that of (p, q). ◀

By Corollary 16 and Proposition 17, we see that a Diophantine logic formula $A : \mathbb{D}_{\mathrm{form}}$ potentially containing inner existential quantifiers and representing the Diophantine relation $\lambda\nu.[\![A]\!]_\nu$ can effectively be reduced to a single Diophantine equation $p' \doteq q' : \mathbb{D}_{\mathrm{single}}(\mathbb{F}_n, \mathsf{V})$ such that $[\![A]\!]_\nu \leftrightarrow \exists\varphi : \mathbb{F}_n \to \mathbb{N}, [\![p']\!]^\varphi_\nu = [\![q']\!]^\varphi_\nu$. Because $\mathbb{F}_n$ is the finite type of n elements, the (higher order) existential quantifier $\exists\varphi$ simply encodes n successive (first order) existential quantifiers. The existential quantifiers that occur deep inside A are *not erased* by the reduction, they are *moved at the outer level* and are ultimately understood as *solvability* for some polynomial equation of which the parameters match the freely occurring variables of A.

5 Remarks on the Implementation of Matiyasevich's Theorems

Matiyasevich's theorem stating that there is a Diophantine description of the exponential relation $x = y^z$ is a masterpiece. It was the last missing step finishing the line of work by Davis, Putnam and Robinson which started with Davis' conjecture in 1953. Already in 1952, Julia Robinson discovered that in order to show the exponential relation Diophantine, it suffices to find a single binary Diophantine relation exhibiting exponential growth [27], a so-called Robinson predicate, i.e. a predicate $J(u, v)$ in two variables s.t. $J(u, v)$ implies $v < u^u$ and for every k there are u, v with $J(u, v)$ and $v > u^k$. Robinson's insight meant the only thing missing to prove what is nowadays called the DPRM theorem, was a *single* polynomial equation capturing *any* freely chosen Robinson predicate. Similar to other famous hard problems of mathematics, the question is easy to state, but from the start of the study of Diophantine equations to the late 1960s, no such relation was known, rendering the problem one of the most baffling questions for mathematicians and computer scientists alike.

In 1970 [19], Yuri Matiyasevich discovered that $v = \mathrm{fib}_{2u}$ is both a Robinson predicate and Diophantine. Here $(\mathrm{fib}_n)_{n\in\mathbb{N}}$ is the well known Fibonacci sequence defined by the second order recurrence relation $\mathrm{fib}_0 = 0$, $\mathrm{fib}_1 = 1$ and $\mathrm{fib}_{n+2} = \mathrm{fib}_{n+1} + \mathrm{fib}_n$. Combined with previous results, this concluded the multi-decades effort to establish the Diophantineness of all recursively enumerable predicates, implying a negative solution to Hilbert's tenth problem. That proof which included the original proof of Matiyasevich [19] was later simplified. For instance, exploiting similar ideas but in the easier context of the solutions of another second order equation – namely Pell's equations $x^2 - (a^2 - 1)y^2 = 1$ with parameter $a > 1$, – Martin Davis [4] gave a standalone proof of the DPRM-theorem where recursively enumerable

predicates are characterised by a variant of μ-recursive functions. In that paper, Davis also provided a proof of the admissibility of bounded universal quantification using the Chinese remainder theorem to encode finite sequences of numbers. There exists more recent and simpler proofs of this admissibility result as well, see e.g. [20].

Before we discuss the mechanisation of the Diophantineness of both the exponential relation and of bounded universal quantification, we want to remark on the difficulty of mechanising the former proof. Both on its own and as a stepping stone towards the negative solution to Hilbert's tenth problem, it is clear that Matiyasevich's theorem was an extremely difficult question which required tremendous intellectual resources to be solved. The mechanisation of a modernised form of the proof, although not trivial, cannot be compared to the difficulty of finding a solution. In particular, the modern proof relies on very mature background theories, lowering the number of possible design choices for the mechanisation. Moreover, very detailed pen and paper accounts of the proof are available, which can be followed closely.

An aspect that is more challenging in mechanisation than on paper are proofs regarding the computability of certain functions. Since paper proofs oftentimes rely on a vague notion of algorithm, most of the reasoning about these algorithms is hand-waved away by computer scientists, relying on the implicit understanding of what is an algorithm. By using a synthetic approach to computability [8, 9, 11], we make the notion of an algorithm precise and thus enable mechanisation, at the same time circumventing the verification of low-level programs.

5.1 Exponential is Diophantine (Theorem 6)

For our mechanised proof, we rely on a more recent account of Matiyasevich's theorem from [21], which, among the many options we considered, seemed the shortest. The proof employs Pell's equation $x^2 - bxy + y^2 = 1$ for $b \geq 2$. We use the second order recurrence relation $\alpha_b(-1) = -1$, $\alpha_b(0) = 0$ and $\alpha_b(n+2) = b\alpha_b(n+1) - \alpha_b(n)$ to describe the set of solutions of Pell's equation by $\big\{(\alpha_b(n), \alpha_b(n+1)) \mid n \in \mathbb{N}\big\}$. The recurrence can be characterised by the following square 2×2 matrix equation:

$$A_b(n) = (B_b)^n \quad \text{with} \quad A_b(n) := \begin{pmatrix} \alpha_b(n+1) & -\alpha_b(n) \\ \alpha_b(n) & -\alpha_b(n-1) \end{pmatrix} \quad \text{and} \quad B_b := \begin{pmatrix} b & -1 \\ 1 & 0 \end{pmatrix}$$

Then, studying the properties of the sequence $n \mapsto \alpha_b(n)$ in $\mathbb{N}$ or $\mathbb{Z}$, one can show that $\alpha_2(n) = n$ and $n \mapsto \alpha_b(n)$ grows exponentially for $b \geq 3$. Studying the properties of the same sequence in $\mathbb{Z}/p\mathbb{Z}$ (for varying values of the modulus p), one can for instance show that $n = \alpha_2(n) \equiv \alpha_b(n) \bmod b{-}2$, which relates n and $\alpha_b(n)$ modulo $(b{-}2)$. With various intricate but elementary results[6] such as e.g. $\alpha_b(k) \mid \alpha_b(m) \leftrightarrow k \mid m$ and $\alpha_b^2(k) \mid \alpha_b(m) \leftrightarrow k\alpha_b(k) \mid m$ (both for $b \geq 2$ and any $k, m \in \mathbb{N}$), one can show that $a, b, c \mapsto 3 < b \wedge a = \alpha_b(c)$ has a Diophantine representation. In our formalisation, we get a Diophantine logic formula of size 490 as a witness (see `dio_rel_alpha_size`).

Once $\alpha_b(n)$ is proven Diophantine, one can recover the exponential relation $x, y, z \mapsto x = y^z$ using the eigenvalue λ of the matrix B_b which satisfies $\lambda^2 - b\lambda - 1 = 0$. By wisely choosing $m = bq - q^2 - 1$, one gets $\lambda \equiv q \bmod m$ and thus, using the corresponding eigenvector, one derives $q\alpha_b(n) - \alpha_b(n-1) \equiv q^n \bmod m$. For a large enough value of m, hence a large enough value[7] of b, this gives a Diophantine representation of q^n. In our code, we get a Diophantine logic formula of size 1689 as a witness (see `dio_rel_expo_size`).

[6] by elementary we certainly do not mean either simple or obvious, but we mean that they only involve standard tools from modular and linear algebra.
[7] the largeness of which is secured using α itself again, but with other input values.

The main libraries which are needed to solve Pell's equation and characterise its solutions are linear algebra (or at least square 2×2 matrices) over commutative rings such as $\mathbb{Z}$ and $\mathbb{Z}/p\mathbb{Z}$, a good library for modular algebra ($\mathbb{Z}/p\mathbb{Z}$), and the binomial theorem over rings. Without the help of the Coq `ring` tactic, such a development would be extremely painful. These libraries are then used again to derive the Diophantine encoding of the exponential.

5.2 Admissibility of Bounded Universal Quantification (Theorem 7)

As hinted earlier, we provide an implementation of the algorithm for the elimination of bounded universal quantification described in [20]. It does not involve the use of a model of computation, hence does not create a chicken-and-egg problem when used for the proof of the DPRM theorem. The technique of [20] uses the exponential function and thus Theorem 6 (a lot), and a combination of arithmetic and bitwise operations over $\mathbb{N}$ through base 2 and base 2^q representations of natural numbers.

The Diophantine admissibility of bitwise operations over $\mathbb{N}$ is based on the relation stating that every bit of a is lower or equal than the corresponding bit in b and denoted $a \preccurlyeq b$. The equation $a \preccurlyeq b \leftrightarrow \mathcal{C}^a_b$ is odd (where $\mathcal{C}^a_b$ denotes the binomial coefficient) gives a Diophantine representation for $a \preccurlyeq b$ and then bitwise operators are derived from $\preccurlyeq$ in combination with regular addition $+$, in particular, the *digit by digit AND* operation called "projection." To obtain that $a \preccurlyeq b$ holds if and only if $\mathcal{C}^a_b \equiv 1 \bmod 2$, we prove Lucas' theorem [18] which allows for the computation of the binomial coefficient in base p. It states that $\mathcal{C}^a_b \equiv \mathcal{C}^{a_n}_{b_n} \times \cdots \times \mathcal{C}^{a_0}_{b_0} \bmod p$ holds when p is prime and $a = a_n p^n + \cdots + a_0$ and $b = b_n p^n + \cdots + b_0$ are the respective base p representations of a and b.[8] A Diophantine representation of the binomial coefficient can be obtained via the binomial theorem: $\mathcal{C}^k_n$ is the k-th digit of the development of $(1+q)^n = \sum_{i=0}^n \mathcal{C}^i_n q^i$ in base $q = 2^{n+1}$. This gives a Diophantine representation using Theorem 6 and the relation `is_digit` defined for Lemma 8.

The rest of the admissibility proof for bounded universal quantification $\forall i,\, i < n \to A$ is a very nice encoding of vectors of natural numbers of type $\mathbb{N}^n$ into natural numbers $\mathbb{N}$ such that regular addition $+$ (resp. multiplication $\times$) somehow performs parallel/simultaneous additions (resp. multiplications) on the encoded vectors. More precisely, a vector $(a_1, \ldots, a_n) \in [0, 2^q - 1]^n$ of natural numbers is encoded as the "cipher" $a_1 r^2 + a_2 r^4 + a_3 r^8 + \cdots + a_n r^{2^n}$ with $r = 2^{4q}$. In these *sparse ciphers,* only the digits occurring at r^{2^i} are non-zero. We remark that none of the parameters, including n or q, are constant in the encoding.

Besides a low-level inductive proof of Lucas' theorem, the essential library for the removal of bounded universal quantification consists of tools to manipulate the type $\mathbb{N}$ simultaneously and smoothly both as (a) usual natural numbers and (b) sparse base $r = 2^{4q}$ encodings of vectors of natural numbers in $[0, 2^q - 1]$. Notice that r is defined as $r = 2^{2q}$ in [20] but we favour the alternative choice $r = 2^{4q}$ which allows for an easier soundness proof for vector multiplication because there is no need to manage for digit overflows (see Appendix D).

A significant step in the Diophantine encoding of $+$ and $\times$ on $\mathbb{N}^n$ is the Diophantine encoding of $u = \sum_{i=1}^n r^{2^i}$ and $u_1 = \sum_{i=2}^{n+1} r^{2^i}$ as the ciphers of the constant vectors $(1, \ldots, 1) \in \mathbb{N}^n$ and $(0, 1, \ldots, 1) \in \mathbb{N}^{n+1}$ respectively, obtained by masking u^2 with $w = \sum_{i=0}^{2^{n+1}} r^i$ and $2w$.

Finally, it should be noted that prior to the elimination of the quantifier in $\forall i,\, i < n \to A$, the Diophantine formula A is first normalised into a conjunction of elementary constraints using Theorem 10, and then the elimination is performed on that list of elementary constraints, encoding e.g. $v_0 \doteq v_1 \dot{+} v_2$ and $v_0 \doteq v_1 \dot{\times} v_2$ with their respective sparse cipher counterparts.

[8] With the usual convention that $\mathcal{C}^a_b = 0$ when $a > b$.

6 Minsky Machines Reduce to FRACTRAN

In previous work, we have reduced the halting problem for Turing machines to PCP [8] and on to a specialised halting problem for Minsky machines [11] in Coq. The specialised halting problem asked whether a machine on a given input halts in a configuration with all registers containing zeros. In order to define Minsky machine recognisability, we consider a general halting problem which allows *any* final configuration. The adaption of the formal proofs reducing PCP via binary stack machines to Minsky machines is quite straightforward and reuses the certified compiler for low-level languages defined in [11].

We first show that one can remove self loops from Minsky machines, i.e. instructions which jump to their own location, using the compositional reasoning techniques developed in [11]. We then formalise the FRACTRAN language [2] and show how the halting problem for Minsky machines can be encoded into the halting problem for FRACTRAN programs. While the verification of Minsky machines can be complex and needs preliminary thoughts on compositional reasoning, the translation from Minsky machines to FRACTRAN is elementary and needs no heavy machinery.

6.1 Minsky Machines

We employ Minsky machines [23] with instructions $\iota : \mathsf{I}_n ::= \texttt{INC}\ (\alpha : \mathbb{F}_n) \mid \texttt{DEC}\ (\alpha : \mathbb{F}_n)\ (p : \mathbb{N})$. A Minsky machine with n registers is a sequence of consecutively indexed instructions $s : \iota_0;\ \ldots\ s+k : \iota_k$; represented as a pair $(s : \mathbb{N}, [\iota_0; \ldots; \iota_k] : \mathbb{L}\,\mathsf{I}_n)$. Its state $(i, \vec{v})$ is a program counter (PC) value $i : \mathbb{N}$ and a vector of values for registers $\vec{v} : \mathbb{N}^n$. $\texttt{INC}\ \alpha$ increases the value of register α and the PC by one. $\texttt{DEC}\ \alpha\ p$ decreases the value of register α by one if that is possible and increases the PC, or, if the register is already 0, jumps to PC value p. Given a Minsky machine (s, P), we write $(s, P) /\!/_M (i_1, \vec{v}_1) \succ^n (i_2, \vec{v}_2)$ when (s, P) transforms state $(i_1, \vec{v}_1)$ into $(i_2, \vec{v}_2)$ in n steps of computation. For (s, P) to do a step in state $(i, \vec{v})$ the instruction at label i in (s, P) is considered. When a label i is outside of the code of (s, P) we write $\texttt{out}\ i\ (s, P)$ and in that case (and only that case), no computation step can occur. We define the halting problem for Minsky Machines as

$$\mathsf{MM}\big(n : \mathbb{N}, P : \mathbb{L}\,\mathsf{I}_n, \vec{v} : \mathbb{N}^n\big) := (1, P) /\!/_M (1, \vec{v})\downarrow$$

where $(s, P)\ /\!/_M\ (i, \vec{v})\downarrow\ :=\ \exists n\, j\, \vec{w}, (s, P)\ /\!/_M\ (i, \vec{v})\ \succ^n\ (j, \vec{w}) \wedge \texttt{out}\ j\ (s, P)$, meaning that the machine (s, P) has a terminating computation starting at state $(i, \vec{v})$. We refer to [11] for a more in-depth formal description of those counter machines. Note that the halting problem defined there is more specific than the problem MM above defined but both are proved undecidable in our library.

We say that a machine *has a self-loop* if it contains an instruction of the form $i : \texttt{DEC}\ \alpha\ i$, i.e. jumps to itself in case the register α has value 0, leading necessarily to non-termination. For every machine P with self-loops, we can construct an equivalent machine Q using one additional register α_0 with constant value 0, which has the same behaviour but no self-loops. Since the effect of a self loop $i : \texttt{DEC}\ \alpha\ i$ is either decrement and move to the next instruction at $i+1$ if $\alpha > 0$ or else enter in a forever loop at i, it is easily simulated by a jump to a length-2 cycle, i.e. replacing $i : \texttt{DEC}\ \alpha\ i$ with $i : \texttt{DEC}\ \alpha\ j$ and adding $j : \texttt{DEC}\ \alpha_0\ (j+1); j+1 : \texttt{DEC}\ \alpha_0\ j$ somewhere near the end of the program.

▶ **Theorem 18.** *Given a Minsky machine P with n registers one can compute a machine Q with $1{+}n$ registers and no self loops s.t. for any $\vec{v}$, $(1, P) /\!/_M (1, \vec{v})\downarrow\ \leftrightarrow\ (1, Q) /\!/_M (1, 0 :: \vec{v})\downarrow$.*

Proof. We explain how any Minsky machine $(1,P)$ with n registers can be transformed into an equivalent one that uses an extra 0 valued spare register $\alpha_0 = 0 \in \mathbb{F}_{1+n}$ and avoids self loops. Let k be the length of P and let P' be the Minsky machine with $1+n$ registers defined by performing a 1-1 replacement of instructions of $(1,P)$:

- instructions of the form $i : \texttt{INC}\ \alpha$ are replaced by $i : \texttt{INC}\ (1+\alpha)$;
- self loops $i : \texttt{DEC}\ \alpha\ i$ are replaced by $i : \texttt{DEC}\ (1+\alpha)\ (2+k)$;
- proper inside jumps $i : \texttt{DEC}\ \alpha\ j$ for $i \neq j$ and $1 \leq j \leq k$ are replaced by $i : \texttt{DEC}\ (1+\alpha)\ j$;
- and outside jumps $i : \texttt{DEC}\ \alpha\ j$ for $j = 0 \vee k < j$ are replaced by $i : \texttt{DEC}\ (1+\alpha)\ 0$.

Then we define $Q := P' \mathbin{+\!\!+} [\texttt{DEC}\ \alpha_0\ 0; \texttt{DEC}\ \alpha_0\ (3+k); \texttt{DEC}\ \alpha_0\ (2+k)]$. Notice that P' is immediately followed $\texttt{DEC}\ \alpha_0\ 0$, i.e. by an unconditional jump to 0 (because α_0 has value 0), and that $(1,Q)$ ends with the length-2 cycle composed of $2+k : \texttt{DEC}\ \alpha_0\ (3+k); 3+k : \texttt{DEC}\ \alpha_0\ (2+k)$. We show that $(1,Q)$ is a program without self loops (obvious) that satisfies the required simulation equivalence. Indeed, self loops are replaced by jumps to the length-2 cycle that uses the unmodified register α_0 to loop forever. One should just be careful that the outside jumps of $(1,P)$ do not accidentally fall into that cycle and this is why we redirect them all to PC value 0. ◀

A predicate $R : \mathbb{N}^n \to \mathbb{P}$ *is MM-recognisable* if there exist $m : \mathbb{N}$ and a Minsky machine $P : \mathbb{L}\,\mathsf{I}_{n+m}$ of $(n+m)$ registers such that for any $\vec{v} : \mathbb{N}^n$ we have $R\ \vec{v} \leftrightarrow (1,P) \mathbin{/\!/_M} (1, \vec{v} \mathbin{+\!\!+} \vec{0})\downarrow$. The last m registers serve as spare registers during the computation. Notice that not allowing for spare registers would make e.g. the empty predicate un-recognisable[9] It is possible to limit the number of (spare) registers but that question is not essential in our development.

6.2 FRACTRAN

We formalise the language FRACTRAN, introduced as a universal programming language for arithmetic by Conway [2]. A FRACTRAN program Q consists of a list of positive fractions $[p_1/q_1; \ldots; p_n/q_n]$. The current state of a FRACTRAN program is just a natural number s. The first fraction p_i/q_i in Q such that $s.(p_i/q_i)$ is still integral determines the successor state, which then is $s.(p_i/q_i)$. If there is no such fraction in Q, the program terminates.

We make this precise inductively for Q being a list of fractions $p/q : \mathbb{N} \times \mathbb{N}$:

$$\frac{q.y = p.x}{(p/q :: Q) \mathbin{/\!/_F} x \succ y} \qquad\qquad \frac{q \nmid p.x \qquad Q \mathbin{/\!/_F} x \succ y}{(p/q :: Q) \mathbin{/\!/_F} x \succ y}$$

i.e. at state x the first fraction p/q in Q where q divides $p.x$ is used, and x is multiplied by p and divided by q. For instance, the FRACTRAN program $[5/7; 2/1]$ runs forever when starting from state 7, producing the sequence $5 = 7.(5/7)$, $10 = 5.(2/1)$, $20 = 10.(2/1)$ ...[10]

We say that a FRACTRAN program $Q = [p_1/q_1; \ldots; p_n/q_n]$ is *regular* if none of its denominators is 0, i.e. if $q_1 \neq 0, \ldots, q_n \neq 0$. For a FRACTRAN program $Q : \mathbb{L}\,(\mathbb{N} \times \mathbb{N})$ and $s : \mathbb{N}$, we define the decision problem as the question “does Q halt when starting from s”:

$$\mathsf{FRACTRAN}(Q,s) := Q \mathbin{/\!/_F} s\downarrow \quad \text{with } Q \mathbin{/\!/_F} s\downarrow := \exists x,\, Q \mathbin{/\!/_F} s \succ^* x \wedge \forall y,\, \neg\, Q \mathbin{/\!/_F} x \succ y$$

Following [2], we now show how (regular) FRACTRAN halting can be used to simulate Minsky machines halting. The idea is to use a simple Gödel encoding of the states of a Minsky

[9] For any Minsky machine $(1,P)$, if it starts on large enough register values, for instance if they are all greater than the length of P, then no jump can occur and the machine terminates after its last instruction executes. Such unfortunate behavior can be circumvented with a 0-valued spare register.

[10] No FRACTRAN program can ever stop when it contains a fraction having an integer value like 2/1.

machine. We first fix two infinite sequences of prime numbers $\mathfrak{p}_0, \mathfrak{p}_1, \ldots$ and $\mathfrak{q}_0, \mathfrak{q}_1, \ldots$ all distinct from each other. We define the encoding of n-register Minsky machine states as $\overline{(i,\vec{v})} := \mathfrak{p}_i \mathfrak{q}_0^{x_0} \ldots \mathfrak{q}_{n-1}^{x_{n-1}}$ where $\vec{v} = [x_0, \ldots, x_{n-1}]$:

- To simulate the step semantics of Minsky machines for $i : \texttt{INC}\ \alpha$, we divide the encoded state by $\mathfrak{p}_i$ and multiply by $\mathfrak{p}_{i+1}$ for the change in PC value, and increment the register α by multiplying with $\mathfrak{q}_\alpha$, hence we add the fraction $\mathfrak{p}_{i+1}\mathfrak{q}_\alpha/\mathfrak{p}_i$;
- To simulate $i : \texttt{DEC}\ \alpha\ j$ when $\vec{v}[\alpha] = 1 + n$ we divide by $\mathfrak{p}_i$, multiply by $\mathfrak{p}_{i+1}$ and decrease register α by dividing by $\mathfrak{q}_\alpha$, hence we add the fraction $\mathfrak{p}_{i+1}/\mathfrak{p}_i\mathfrak{q}_\alpha$;
- To simulate $i : \texttt{DEC}\ \alpha\ j$ when $\vec{v}[\alpha] = 0$ we divide by $\mathfrak{p}_i$ and multiply by $\mathfrak{p}_j$. To make sure that this is only executed when the previous rule does not apply, we add the fraction $\mathfrak{p}_j/\mathfrak{p}_i$ *after* the fraction $\mathfrak{p}_{i+1}/\mathfrak{p}_i\mathfrak{q}_\alpha$.

In short, we define the encoding of labelled instructions and then programs as

$$\overline{(i, \texttt{INC}\ \alpha)} := [\mathfrak{p}_{i+1}\mathfrak{q}_\alpha/\mathfrak{p}_i] \qquad \overline{(i, [\iota_0; \ldots; \iota_k])} := \overline{(i,\iota_0)} + \cdots + \overline{(i+k,\iota_k)}.$$
$$\overline{(i, \texttt{DEC}\ \alpha\ j)} := [\mathfrak{p}_{i+1}/\mathfrak{p}_i\mathfrak{q}_\alpha; \mathfrak{p}_j/\mathfrak{p}_i]$$

Notice that we only produce regular programs and that a self loop like $i : \texttt{DEC}\ \alpha\ i$, jumping on itself when $\vec{v}[\alpha] = 0$, will generate the fraction $\mathfrak{p}_i/\mathfrak{p}_i$ potentially capturing any state $\overline{(j,\vec{v})}$ even when $j \neq i$. So this encoding does not work on Minsky machines containing self loops.

▶ **Lemma 19.** *If $(1,P)$ has no self loops then $(1,P) /\!/_M (1,\vec{v})\downarrow \leftrightarrow \overline{(1,P)} /\!/_F \overline{(1,\vec{v})}\downarrow$.*

Proof. Let (i,P) be a Minsky machine with no self loops. We show that the simulation of (i,P) by $\overline{(i,P)}$ is 1-1, i.e. each step is simulated by one step. We first show the forward simulation, i.e. that $(i,P) /\!/_M (i_1,\vec{v}_1) \succ (i_2,\vec{v}_2)$ entails $\overline{(i,P)} /\!/_F \overline{(i_1,\vec{v}_1)} \succ \overline{(i_2,\vec{v}_2)}$, by case analysis. Conversely we show that if $\overline{(i,P)} /\!/_F \overline{(i_1,\vec{v}_1)} \succ st$ holds then $st = \overline{(i_2,\vec{v}_2)}$ for some $(i_2,\vec{v}_2)$ such that $(i,P) /\!/_M (i_1,\vec{v}_1) \succ (i_2,\vec{v}_2)$. Backward simulation involves the totality of MM one step semantics and the determinism of regular FRACTRAN one step semantics combined with the forward simulation.

Using these two simulation results, the desired equivalence follows by induction on the length of terminating computations. ◀

▶ **Theorem 20.** *For any n-register Minsky machine P one can compute a regular FRACTRAN program Q s.t. $(1,P) /\!/_M (1,[x_1;\ldots;x_n])\downarrow \leftrightarrow Q /\!/_F \mathfrak{p}_1\mathfrak{q}_1^{x_1}\ldots\mathfrak{q}_n^{x_n}\downarrow$ holds for any $x_1,\ldots,x_n$.*

Proof. Using Theorem 18, we first compute a Minsky machine $(1,P_1)$ equivalent to $(1,P)$ but with one extra 0-valued spare register and no self loops. Then we apply Lemma 19 to $(1,P_1)$ and let $Q := \overline{(1,P_1)}$. The program Q is obviously regular and given $\vec{v} = [x_1;\ldots;x_n]$, the encoding of the starting state $(1, 0 :: \vec{v})$ for $(1,P_1)$ is $\mathfrak{p}_1\mathfrak{q}_0^0\mathfrak{q}_1^{x_1}\ldots\mathfrak{q}_n^{x_n}$ hence the result. ◀

This gives us a formal constructive proof that (regular) FRACTRAN is Turing complete as a model of computation and is consequently undecidable.

▶ **Corollary 21.** *Halt reduces to FRACTRAN.*

Proof. Theorem 20 gives us a reduction from MM to FRACTRAN which can be combined with the reduction of Halt to PCP from [8] and a slight modification of PCP to MM from [11]. ◀

7 Diophantine Encoding of FRACTRAN

We show that a single step of FRACTRAN computation is a Diophantine relation.

▶ **Lemma 22.** *For any FRACTRAN program* $Q : \mathbb{L}(\mathbb{N}\times\mathbb{N})$ *and any* $f, g : (\mathsf{V} \to \mathbb{N}) \to \mathbb{N}$, *there is a map* $\mathbb{D}_{\mathrm{P}}\, f \to \mathbb{D}_{\mathrm{P}}\, g \to \mathbb{D}_{\mathrm{R}}\,(\lambda\nu.\, Q \mathbin{/\!/_F} f_\nu \succ g_\nu)$.

Proof. The map is built by induction on Q. If $Q = [\,]$, then we show $[\,] \mathbin{/\!/_F} f_\nu \succ g_\nu \leftrightarrow \texttt{False}$, and thus $\mathbb{D}_{\mathrm{R}}\,(\lambda\nu.Q \mathbin{/\!/_F} f_\nu \succ g_\nu)$ by map 4 of Lemma 4 and Proposition 3. If Q is a composed list $Q = p/q :: Q'$, then we show the equivalence

$$(p/q :: Q') \mathbin{/\!/_F} f_\nu \succ g_\nu \;\leftrightarrow\; q.g_\nu = p.f_\nu \vee q \nmid (p.f_\nu) \wedge Q' \mathbin{/\!/_F} f_\nu \succ g_\nu$$

and we derive $\mathbb{D}_{\mathrm{R}}\,(\lambda\nu.Q \mathbin{/\!/_F} f_\nu \succ g_\nu)$ by map 4 of Lemma 4, Proposition 5 and the induction hypothesis, these last steps being automated by the `dio_rel_auto` tactic. ◀

In addition, the "Q has terminated at x" predicate is Diophantine for any FRACTRAN program Q. The proof is similar to the previous one:

▶ **Lemma 23.** *For any FRACTRAN program* $Q : \mathbb{L}(\mathbb{N}\times\mathbb{N})$ *and any* $f : (\mathsf{V} \to \mathbb{N}) \to \mathbb{N}$, *there is a map* $\mathbb{D}_{\mathrm{P}}\, f \to \mathbb{D}_{\mathrm{R}}\,(\lambda\nu.\forall y, \neg\, Q \mathbin{/\!/_F} f_\nu \succ y)$.

Proof. The map $\forall f, \mathbb{D}_{\mathrm{P}}\, f \to \mathbb{D}_{\mathrm{R}}\,(\lambda\nu.\forall y, \neg\, Q \mathbin{/\!/_F} f_\nu \succ y)$ is built by induction on Q. If $Q = [\,]$, then we show $(\forall y, \neg\, [\,] \mathbin{/\!/_F} f_\nu \succ y) \leftrightarrow \texttt{True}$, and thus $\mathbb{D}_{\mathrm{R}}\,(\lambda\nu.\forall y, \neg\, Q \mathbin{/\!/_F} f_\nu \succ y)$ by map 4 of Lemma 4 and Proposition 3. If $Q = p/q :: Q'$, then we show the equivalence $\forall y, \neg\, Q \mathbin{/\!/_F} f_\nu \succ y \leftrightarrow q \nmid (p.f_\nu) \wedge \forall y, \neg\, Q' \mathbin{/\!/_F} f_\nu \succ y$ and we get $\mathbb{D}_{\mathrm{R}}\,(\lambda\nu.\forall y, \neg\, Q \mathbin{/\!/_F} f_\nu \succ y)$ by map 4 of Lemma 4, Proposition 5 and the induction hypothesis. ◀

We can now deduce a core result of the paper which states that FRACTRAN programs have Diophantine termination predicates.

▶ **Theorem 24.** *If Q is a FRACTRAN program and* $\mathbb{D}_{\mathrm{P}}\, f$ *then* $\mathbb{D}_{\mathrm{R}}\,(\lambda\nu.\, Q \mathbin{/\!/_F} f_\nu \downarrow)$.

Proof. By definition we have $Q \mathbin{/\!/_F} f_\nu \downarrow \leftrightarrow \exists x\,(Q \mathbin{/\!/_F} f_\nu \succ^* x \wedge \forall y,\, \neg\, Q \mathbin{/\!/_F} x \succ y)$ and hence we obtain the claim using Theorem 9 in conjunction with Lemma 22 and Lemma 23. ◀

We conclude with the undecidability of Hilbert's tenth problem by a reduction chain starting from the Halting problem for single tape Turing machines:

▶ **Theorem 25** (Hilbert's tenth problem). *We have the following reduction chain*

$$\mathit{Halt} \preceq \mathit{PCP} \preceq \mathit{MM} \preceq \mathit{FRACTRAN} \preceq \mathit{DIO_FORM} \preceq \mathit{DIO_ELEM} \preceq \mathit{DIO_SINGLE} \preceq \mathit{H10}$$

and as a consequence, H10 is undecidable.

Proof. The proof combines the previous results like Theorems 20 and 24 and Corollary 16. ◀

8 The Davis-Putnam-Robinson-Matiyasevich Theorem

We conclude the paper with a proof of the DPRM theorem stating that recursively enumerable predicates are Diophantine. Here we assume that the informal notion of "recursive enumerability" can be characterised by Minsky machines recognisability (see Section 6.1).

▶ **Lemma 26.** *For FRACTRAN programs Q we have* $\mathbb{D}_{\mathrm{R}}\left(\lambda\nu.\, Q \mathbin{/\!/_F} \mathfrak{p}_1 \mathfrak{q}_1^{\nu\, x_0} \ldots \mathfrak{q}_n^{\nu\, x_{n-1}} \downarrow\right)$.

Proof. By induction on $n : \mathbb{N}$, we show $\forall f$, $\mathbb{D}_\mathrm{P}\, f \to \mathbb{D}_\mathrm{R}\,(\lambda\nu. f_\nu = \mathfrak{p}_1 \mathfrak{q}_1^{\nu\, x_0} \dots \mathfrak{q}_n^{\nu\, x_{n-1}})$. Notice that $\mathfrak{p}_1$ and the $\mathfrak{q}_i$'s are hard-coded[11] in the Diophantine representation but we of course use Theorem 6. Then we end the proof by a combination with Theorem 24. ◀

▶ **Theorem 27** (DPRM). *Any MM-recognisable relation $R : \mathbb{N}^n \to \mathbb{P}$ is Diophantine: one can compute a single Diophantine equation $p \doteq q : \mathbb{D}_{\text{single}}(\mathbb{F}_m, \mathbb{F}_n)$ with n parameters and m variables s.t. $\forall \vec{v} : \mathbb{N}^n,\ R\ \vec{v} \leftrightarrow \exists \vec{w} : \mathbb{N}^m,\ [\![p]\!]_{\vec{v}}^{\vec{w}} = [\![q]\!]_{\vec{v}}^{\vec{w}}$.*[12]

Proof. By definition, $R : \mathbb{N}^n \to \mathbb{P}$ is recognised by some Minsky machine P with $(n+m)$ registers, i.e. $R\ \vec{v} \leftrightarrow (1, P)\ /\!\!/_M\ (1, \vec{v} +\!\!+ \vec{0})\downarrow$. By Theorem 20, we compute a FRACTRAN program Q s.t. $(1, P)\ /\!\!/_M\ (1, [v_1; \dots; v_n; w_1; \dots; w_m])\downarrow \leftrightarrow Q\ /\!\!/_F\ \mathfrak{p}_1 \mathfrak{q}_1^{v_1} \dots \mathfrak{q}_n^{v_n} \mathfrak{q}_{n+1}^{w_1} \dots \mathfrak{q}_{n+m}^{w_m}\downarrow$.

Hence we deduce $R\ [v_1; \dots; v_n] \leftrightarrow Q\ /\!\!/_F\ \mathfrak{p}_1 \mathfrak{q}_1^{v_1} \dots \mathfrak{q}_n^{v_n}\downarrow$. As a consequence, the relation $\lambda\nu. R\ [\nu\, x_0; \dots; \nu\, x_{n-1}]$ is Diophantine by Lemma 26. By Corollary 16, there is a Diophantine equation $p \doteq q : \mathbb{D}_{\text{single}}(\mathbb{N}, \mathsf{V})$ such that $R\ [\nu\, x_0; \dots; \nu\, x_{n-1}] \leftrightarrow \exists\varphi,\ [\![p]\!]_\nu^\varphi = [\![q]\!]_\nu^\varphi$. Notice that the value $\nu\, x_i$ of any parameter of $p \doteq q$ greater than x_n does not influence solvability.

Now let m be an upper bound of the number of (existentially quantified) variables in $p \doteq q$. We injectively map those variables in $\mathbb{F}_m$ and we project the parameters of $p \doteq q$ onto $\mathbb{F}_n$ by replacing every parameter greater than x_n with the 0 constant. We get a Diophantine equation $p' \doteq q' : \mathbb{D}_{\text{single}}(\mathbb{F}_m, \mathbb{F}_n)$ of which the solvability at $\vec{v}$ is equivalent to $R\ \vec{v}$. ◀

9 Related and Future Work

Regarding formalisations of Hilbert's tenth problem, there are various unfinished and preliminary results in different proof assistants: Carneiro [1] formalises Matiyasevich's theorem (Diophantineness of exponentiation) in Lean, but does not consider computational models or the DPRM theorem. Pąk formalises results regarding Pell's equation [24] and proves that Diophantine sets are closed under union and intersection [25], both as parts of the Mizar Mathematical Library. Stock et al. [28] report on an unfinished formalisation of the DPRM theorem in Isabelle based on [21]. They cover some parts of the proof, but acknowledge for important missing results like Lucas' or "Kummer's theorem" and a "formalisation of a register machine." Moreover, none of the cited reports considers the computability of the reductions involved or the verification of a universal machine in the chosen model of computation yet, one of them being a necessary proof goal for an actual undecidability result in the classical meta-theories of Isabelle/HOL and Mizar.

Regarding undecidability proofs in type theory, Forster, Heiter, and Smolka [8] reduce the halting problem of Turing machines to PCP. Forster and Larchey-Wendling [11] reduce PCP to provability in linear logic via the halting problem of Minsky machines, which we build on. Forster, Kirst and Smolka develop the notion of synthetic undecidability in Coq and prove the undecidability of various notions in first-order logic [9].

In future developments, we want to connect our work to the formalisation of the recent simplified undecidability proof for System F inhabitation by Dudenhefner and Rehof [7], which builds on elementary Diophantine constraints. The undecidability of second-order unification shown by Goldfarb [14] is also by reduction from elementary Diophantine constraints. We want to formalise his proof as an addition to our library of undecidable problems.

---

[11] Which means we do not need to encode the algorithm that actually computes them.

[12] In the notation $[\![p]\!]_{\vec{v}}^{\vec{w}}$ we abusively identify the vector $\vec{v} : \mathbb{N}^n$ (resp. $\vec{w} : \mathbb{N}^m$) with the valuation $\lambda(i : \mathbb{F}_n).\vec{v}[i]$ (resp. $\lambda(j : \mathbb{F}_m).\vec{w}[j]$) that accesses the components of the vector $\vec{v}$ (resp. $\vec{w}$).

In the present paper, we prove that every MM-recognisable problem is Diophantine. This result can be extended to an equivalence, and furthermore to other formalised models of computation like μ-recursive functions [17], Turing machines, or the untyped λ-calculus [12].

References

1. Mario Carneiro. A Lean formalization of Matiyasevič's theorem, 2018. `arXiv:1802.01795`.
2. John H. Conway. *FRACTRAN: A Simple Universal Programming Language for Arithmetic*, pages 4–26. Springer New York, New York, NY, 1987.
3. Martin Davis. Arithmetical problems and recursively enumerable predicates 1. *The Journal of Symbolic Logic*, 18(1):33–41, 1953.
4. Martin Davis. Hilbert's Tenth Problem is Unsolvable. *The American Mathematical Monthly*, 80(3):233–269, 1973.
5. Martin Davis and Hilary Putnam. *A computational proof procedure; Axioms for number theory; Research on Hilbert's Tenth Problem.* Air Force Office of Scientific Research, Air Research and Development, 1959.
6. Martin Davis, Hilary Putnam, and Julia Robinson. The decision problem for exponential Diophantine equations. *Annals of Mathematics*, pages 425–436, 1961.
7. Andrej Dudenhefner and Jakob Rehof. A Simpler Undecidability Proof for System F Inhabitation. *TYPES 2018*, 2018.
8. Yannick Forster, Edith Heiter, and Gert Smolka. Verification of PCP-Related Computational Reductions in Coq. In *ITP 2018*, pages 253–269. Springer, 2018.
9. Yannick Forster, Dominik Kirst, and Gert Smolka. On Synthetic Undecidability in Coq, with an Application to the Entscheidungsproblem. In *CPP 2019*, pages 38–51, 2019.
10. Yannick Forster and Dominique Larchey-Wendling. Towards a library of formalised undecidable problems in Coq: The undecidability of intuitionistic linear logic. *Workshop on Syntax and Semantics of Low-level Languages, Oxford*, 2018.
11. Yannick Forster and Dominique Larchey-Wendling. Certified Undecidability of Intuitionistic Linear Logic via Binary Stack Machines and Minsky Machines. In *CPP 2019*, pages 104–117. ACM, 2019. `doi:10.1145/3293880.3294096`.
12. Yannick Forster and Gert Smolka. Weak Call-By-Value Lambda Calculus as a Model of Computation in Coq. In *ITP 2018*, pages 189–206. Springer, 2017.
13. Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für mathematik und physik*, 38(1):173–198, 1931.
14. Warren D. Goldfarb. The undecidability of the secondorder unification problem. *Theoretical Computer Science*, 13:225–230, 1981.
15. David Hilbert. Mathematical problems. *Bulletin of the American Mathematical Society*, 8(10):437–479, 1902.
16. J. P. Jones and Y. V. Matijasevič. Register Machine Proof of the Theorem on Exponential Diophantine Representation of Enumerable Sets. *J. Symb. Log.*, 49(3):818–829, 1984. `doi:10.2307/2274135`.
17. Dominique Larchey-Wendling. Typing Total Recursive Functions in Coq. In *ITP 2017*, pages 371–388. Springer, 2017.
18. Edouard Lucas. Théorie des Fonctions Numériques Simplement Périodiques. [Continued]. *American Journal of Mathematics*, 1(3):197–240, 1878.
19. Yuri V. Matijasevič. Enumerable sets are Diophantine. In *Soviet Mathematics: Doklady*, volume 11, pages 354–357, 1970.
20. Yuri V. Matiyasevich. A new technique for obtaining Diophantine representations via elimination of bounded universal quantifiers. *J. Math. Sci.*, 87(1):3228–3233, 1997.
21. Yuri V. Matiyasevich. On Hilbert's Tenth Problem. Expository Lectures 1, Pacific Institute for the Mathematical Sciences, University of Calgary, February 2000. URL: `http://www.mathtube.org/sites/default/files/lecture-notes/Matiyasevich.pdf`.

22 Yuri V. Matiyasevich. Martin Davis and Hilbert's Tenth Problem. In *Martin Davis on Computability, Computational Logic, and Mathematical Foundations*. Springer, 2016.

23 Marvin L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., 1967.

24 Karol Pąk. The Matiyasevich Theorem. Preliminaries. *Formalized Mathematics*, 25(4):315–322, 2017.

25 Karol Pąk. Diophantine sets. Preliminaries. *Formalized Mathematics*, 26(1):81–90, 2018.

26 Emil L. Post. Recursively enumerable sets of positive integers and their decision problems. *bulletin of the American Mathematical Society*, 50(5):284–316, 1944.

27 Julia Robinson. Existential definability in arithmetic. *Transactions of the American Mathematical Society*, 72(3):437–449, 1952.

28 Benedikt Stock et al. Hilbert Meets Isabelle: Formalisation of the DPRM Theorem in Isabelle. *Isabelle Workshop 2018*, 2018. `doi:10.29007/3q4s`.

29 The Coq Proof Assistant. `http://coq.inria.fr`, 2019.

A Some numerical Details about the Coq Code Contents

We give a detailed overview of the structure of the code corresponding to the results presented in this paper, and which was contributed to our Coq library of undecidable problems. The following *lines of code (loc)* measurements combine both definitions and proof scripts but do not account for comments. Notice that there are more files in the whole library than those needed to actually cover H10, but here, we only present the latter. In total, we contribute 12k loc to our undecidability project, 4k being additions to its shared libraries as extensions of the Coq standard library.

Concerning the multi-purpose shared libraries in Shared/Libs/DLW/Utils:

- we implemented finitary sums/products (over monoids) up to the binomial theorem (Newton) over non-commutative rings in sums.v and binomial.v for a total of 550 loc;
- we implemented bitwise operations over $\mathbb{N}$, both a lists of bits in bool_list.v and Peano `nat` in bool_nat.v for a total of 1700 loc;
- we implemented many results about Euclidean division and Bézout's identity in gcd.v, prime numbers and their unboundedness in prime.v, and base p representations in power_decomp.v for a total of 1200 loc;
- we implemented miscellaneous libraries for the reification of bounded_quantification.v (120 loc), the Pigeon Hole Principle in php.v (350 loc) and iterations of binary relations in rel_iter.v (230 loc).

Concerning the libraries for Minsky machines and FRACTRAN programs:

- by a slight update to the existing code [11], we proved in mm_comp.v that MM-termination (on any state) is undecidable (10 loc). Both the pre-existing result (undecidability of MM-termination on the zero state) and the new result derive from the correctness of the compiler of binary stack machines into Minsky machines;
- we implemented the removal of self-loops in Minsky machines in mm_no_self.v (340 loc);
- we construct two infinite sequences of primes $\mathfrak{p}_i$ and $\mathfrak{q}_i$ in prime_seq.v (240 loc);
- FRACTRAN definitions and basic results occur in fractran_defs.v (310 loc) and the verified compiler from Minsky machines to FRACTRAN occurs in mm_fractran.v (300 loc);

Concerning the libraries for proving Matiyasevich's theorems:

- we implemented a library for modular arithmetic ($\mathbb{Z}/p\mathbb{Z}$) in Zp.v (920 loc);
- we implemented a library for 2×2-matrix computation including exponentiation and determinants in matrix.v (210 loc);
- we implemented an elementary proof of Lucas' theorem in luca.v (290 loc);

- the solution $\alpha_b(n)$ of Pell's equation and its (modular) arithmetic properties up to a proof of its Diophantineness are in alpha.v (1150 loc);
- from $\alpha_b(n)$, we implement the meta-level Diophantine encoding of the exponential in expo_diophantine.v (150 loc);
- we implement the sparse ciphers used in the Diophantine elimination of bounded universal quantification in cipher.v (1450 loc).

Concerning the object-level Diophantine libraries:

- the definition of Diophantine logic and basic results is in dio_logic.v (450 loc);
- the definition of elementary Diophantine constraints and the reduction from Diophantine logic is in dio_elem.v (580 loc);
- the definition of single Diophantine equations and the reduction from elementary Diophantine constraints is in dio_single.v (350 loc);
- we implement the object-level Diophantine encoding of the exponential relation in dio_expo.v (130 loc); but all the work is done in the previously mentioned libraries;
- the object-level Diophantine encoding of bounded universal quantification spans over dio_binary.v, dio_cipher.v and dio_bounded.v (430 loc);
- we derive the object-level Diophantine encoding of the reflexive-transitive closure in dio_rt_closure.v (40 loc);
- we implement the object-level Diophantine encoding of the FRACTRAN termination predicate in fractran_dio.v (110 loc).

To finish, the main undecidability results and the DPRM:

- the undecidability of Minsky machines is in HALT_MM.v (20 loc);
- the reduction from MM to FRACTRAN is in MM_FRACTRAN.v (50 loc);
- the Diophantine encoding of FRACTRAN termination is in FRACTRAN_DIO.v (70 loc);
- the whole reduction chain leading to the undecidability of H10 is in H10.v (60 loc);
- and the DPRM theorem is in DPRM.v (45 loc).

B Atomic Formulæ as Elementary Constraints (Lemma 12)

We complete the postponed part of the proof of Lemma 12. We compute a representation for an atomic logical formula $p \doteq q : \mathbb{D}_{\text{form}}$ in an interval $[u_a, u_{a+n}[\subseteq \mathsf{U}$.[13] We describe the technique on the example of the atomic formula $x_1 \dot{+} (x_2 \dot{\times} 5) \doteq x_3$ which we represent in the interval $[u_0, u_{10}[$. Let us consider the following definitions:

$$\mathcal{E} := \begin{bmatrix} u_0 \doteq u_1 \dot{+} u_2 \ ; \ u_3 \doteq u_1 \dot{+} u_4 \ ; \ u_3 \doteq u_2 \dot{+} u_9 \ ; \\ u_4 \doteq u_5 \dot{+} u_6 \ ; \ u_5 \doteq x_1 \ ; \ u_6 \doteq u_7 \dot{\times} u_8 \ ; \ u_7 \doteq x_2 \ ; \ u_8 \doteq 5 \ ; \\ u_9 \doteq x_3 \end{bmatrix} \qquad \mathfrak{r} := u_0$$

The second line of $\mathcal{E}$ encodes the expression $x_1 \dot{+} (x_2 \dot{\times} 5)$ in $[u_4, u_9[$, and the third line encodes the expression x_3 in $[u_9, u_{10}[$ in a directed way: the values of the u_i's are uniquely determined by the values of the x_i's and the value of u_4 (resp. u_9) is always the same as the value of $x_1 \dot{+} (x_2 \dot{\times} 5)$ (resp. x_3). By "directed" we mean that the encoding is oriented bottom-up by the syntactic tree of sub-expressions: each variable in $[u_4, u_9[$ (resp. $[u_9, u_{10}[$) encodes a sub-expression of $x_1 \dot{+} (x_2 \dot{\times} 5)$ (resp. x_3) and its value is always the same as the value of the corresponding sub-expression.

The first line encodes the identity sign in $x_1 \dot{+} (x_2 \dot{\times} 5) \doteq x_3$. Indeed, whatever the values of u_4 and u_9, the three constraints of the first line give enough freedom (in the choice of

[13] The value of a is an input but the value of n is an output.

u_1, u_2, u_3) to always be satisfiable (requirement 3 of Definition 11). But when the single constraint $u_0 \doteq 0$ is added (because $\mathfrak{r}$ is u_0), then u_1 and u_2 must evaluate to 0 (because of $u_0 \doteq u_1 \dotplus u_2$) and then u_3 must have the same value as both u_4 (because of $u_3 \doteq u_1 \dotplus u_4$) and u_9 (because of $u_3 \doteq u_2 \dotplus u_9$), hence the identity $x_1 \dotplus (x_2 \dot{\times} 5) \doteq x_3$ must be satisfied (requirement 4 of Definition 11).

C Proof of a Convexity Identity (Proposition 13)

We give an elementary arithmetic justification of the result, proof which involves none of the high-level tools of mathematical analysis. We first show the statement

$$\text{for any } a, b : \mathbb{N}, \text{ we have } 2ab \leq a^2 + b^2 \text{ and } 2ab = a^2 + b^2 \leftrightarrow a = b \tag{1}$$

Assuming without loss of generality that $a \leq b$, we can write $b = a + \delta$ with $\delta \in \mathbb{N}$ and then, for $\bowtie \in \{\leq, =\}$ we have $2ab \bowtie a^2 + b^2 \leftrightarrow 2a^2 + 2a\delta \bowtie a^2 + a^2 + 2a\delta + \delta^2 \leftrightarrow 0 \bowtie \delta^2$ hence the desired result.

Then we proceed with the proof of $\sum_{i=1}^n 2p_iq_i = \sum_{i=1}^n p_i^2 + q_i^2 \leftrightarrow p_1 = q_1 \wedge \cdots \wedge p_n = q_n$. The *if case* is obvious so we only describe the *only if case*. If there is $u \in [1, n]$ such that $p_u \neq q_u$ then we have $2p_uq_u < p_u^2 + q_u^2$, and $2p_jq_j \leq p_j^2 + q_j^2$ for all the other $j \in [1, n] - \{u\}$, in all cases by Statement (1). Hence we get $\sum_{i=1}^n 2p_iq_i < \sum_{i=1}^n p_i^2 + q_i^2$ and the identity is not possible. So the only way to get the identity is when $p_u = q_u$ holds for any $i \in [1, n]$. Despite its "classical logic" taste, this argument can easily be transformed into a constructive one by reasoning inductively on n.

D Avoiding Overflows in the Proof of Theorem 7

The section explains why we slightly modified the original proof of the elimination of bounded universal quantification [20] to avoid overflows when multiplying ciphers. Considering Equation (40) of page 3232, we compute the following product of ciphers

$$\sum_{i=1}^n a_i r^{2^i} \times \sum_{i=1}^n b_i r^{2^i} = \sum_{i=1}^n a_i b_i r^{2^{i+1}} + \sum_{1 \leq i < j \leq n} (a_i b_j + a_j b_i) r^{2^i + 2^j}$$

and we remark that $a_ib_j + a_jb_i$ overflows over $r = 2^{2q}$ for e.g. $a_i = a_j = b_i = b_j = 2^q - 1$. This slight overflow makes the implementation of the proof that the right part $\sum_{i<j}(a_ib_j + a_jb_i)r^{2^i+2^j}$ is masked out in Equation (40) significantly harder.

On the other hand, for r alternatively chosen as e.g. $r = 2^{4q}$, the overflow does not occur any more. With this remark, we do not imply that Equation (40) of [20] is incorrect in any way. However, its formal proof is really more complicated when overflows occur and that situation is straightforward to avoid.

The Δ-calculus: Syntax and Types

Luigi Liquori
Université Côte d'Azur, Inria, Sophia Antipolis, France
Luigi.Liquori@inria.fr

Claude Stolze
Université Côte d'Azur, Inria, Sophia Antipolis, France
Claude.Stolze@inria.fr

Abstract

We present the Δ-calculus, an explicitly typed λ-calculus with strong pairs, projections and explicit type coercions. The calculus can be parametrized with different intersection type theories $\mathcal{T}$, e.g. the Coppo-Dezani, the Coppo-Dezani-Sallé, the Coppo-Dezani-Venneri and the Barendregt-Coppo-Dezani ones, producing a family of Δ-calculi with related intersection typed systems. We prove the main properties like Church-Rosser, unicity of type, subject reduction, strong normalization, decidability of type checking and type reconstruction. We state the relationship between the intersection type assignment systems à la Curry and the corresponding intersection typed systems à la Church by means of an essence function translating an explicitly typed Δ-term into a pure λ-term one. We finally translate a Δ-term with type coercions into an equivalent one without them; the translation is proved to be coherent because its essence is the identity. The generic Δ-calculus can be parametrized to take into account other intersection type theories as the ones in the Barendregt et al. book.

2012 ACM Subject Classification Theory of computation → Lambda calculus; Theory of computation → Type theory

Keywords and phrases intersection types, lambda calculus à la Church and à la Curry, proof-functional logics

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.28

Related Version A full version of the paper is available at `https://arxiv.org/abs/1803.09660`.

Funding Work supported by the COST Action CA15123 EUTYPES "The European research network on types for programming and verification".

Acknowledgements We are grateful to Benjamin Pierce, Joe Wells, Furio Honsell, and the anonymous reviewers for the useful comments and remarks.

1 Introduction

Intersection type theories $\mathcal{T}$ were first introduced as a form of ad hoc polymorphism in (pure) λ-calculi à la Curry. The paper by Barendregt, Coppo, and Dezani [4] is a classic reference, while [5] is a definitive reference.

Intersection type assignment systems $\lambda_\cap^\mathcal{T}$ have been well known in the literature for almost 40 years for many reasons: among them, characterization of strongly normalizing λ-terms [5], λ-models [1], automatic type inference [28], type inhabitation [45, 40], type unification [18].

As intersection had its classical development for type assignment systems, many papers tried to find an explicitly typed λ-calculus à la Church corresponding to the original intersection type assignment systems à la Curry. The programming language Forsythe, by Reynolds [41], is probably the first reference, while Pierce's Ph.D. thesis [36] combines also unions, intersections and bounded polymorphism. In [49] intersection types were used as a foundation for typed intermediate languages for optimizing compilers for higher-order

© Luigi Liquori and Claude Stolze;
licensed under Creative Commons License CC-BY
4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019).
Editor: Herman Geuvers; Article No. 28; pp. 28:1–28:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

polymorphic programming languages; implementations of typed programming language featuring intersection (and union) types can be found in SML-CIDRE [15] and in StardustML [19, 20].

Annotating pure λ-terms with intersection types is not simple: a classical example is the difficulty to decorate the bound variable of the explicitly typed polymorphic identity $\lambda x{:}?.x$ such that the type of the identity is $(\sigma \to \sigma) \cap (\tau \to \tau)$: previous attempts showed that the full power of the intersection type discipline can be easily lost.

In this paper, we define and prove the main properties of the Δ-calculus, a generic intersection typed system for an explicitly typed λ-calculus à la Church enriched with strong pairs, denoted by $\langle \Delta_1 , \Delta_2 \rangle$, projections, denoted by $pr_i\, \Delta$, and type coercions, denoted by Δ^σ.

A *strong pair* $\langle \Delta_1 , \Delta_2 \rangle$ is a special kind of cartesian product such that the two parts of a pair satisfy a given property $\mathcal{R}$ on their "essence", that is $\wr \Delta_1 \wr\, \mathcal{R} \,\wr \Delta_2 \wr$.

An *essence* $\wr \Delta \wr$ of a Δ-term is a pure λ-term obtained by erasing type decorations, projections and choosing one of the two elements inside a strong pair. As examples

$$\begin{array}{rcl} \wr \langle \lambda x{:}\sigma \cap \tau.pr_2\, x \,, \lambda x{:}\sigma \cap \tau.pr_1\, x \rangle \wr & = & \lambda x.x \\ \wr \lambda x{:}(\sigma \to \tau) \cap \sigma.(pr_1\, x)(pr_2\, x) \wr & = & \lambda x.x\, x \\ \wr \lambda x{:}\sigma \cap (\tau \cap \rho).\langle \langle pr_1\, x \,, pr_1\, pr_2\, x \rangle \,, pr_2\, pr_2\, x \rangle \wr & = & \lambda x.x \end{array}$$

and so on. Therefore, the essence of a Δ-term is its untyped skeleton: a strong pair $\langle \Delta_1 , \Delta_2 \rangle$ can be typechecked if and only if $\wr \Delta_1 \wr\, \mathcal{R} \,\wr \Delta_2 \wr$ is verified, otherwise the strong pair will be ill-typed. The essence also gives the exact mapping between a term and its typing à la Church and its corresponding term and type assignment à la Curry. Changing the parameters $\mathcal{T}$ and $\mathcal{R}$ results in defining a totally different intersection typed system.

A *type coercion* Δ^τ is a term of type τ whose type-decoration denotes an application of a subsumption rule to the term Δ of type σ such that $\sigma \leqslant_{\mathcal{T}} \tau$: if we omit type coercions, then we lose the uniqueness of type property.

For the purpose of this paper, we study the four well-known intersection type theories $\mathcal{T}$, namely Coppo-Dezani $\mathcal{T}_{\mathrm{CD}}$ [12], Coppo-Dezani-Sallé $\mathcal{T}_{\mathrm{CDS}}$ [13], Coppo-Dezani-Venneri $\mathcal{T}_{\mathrm{CDV}}$ [14] and Barendregt-Coppo-Dezani $\mathcal{T}_{\mathrm{BCD}}$ [4]. We will inspect the above type theories using three equivalence relations $\mathcal{R}$ on pure λ-terms, namely $\equiv$, $=_\beta$, and $=_{\beta\eta}$.

The combination of the above $\mathcal{T}$ and $\mathcal{R}$ allows to define ten meaningful typed systems for the Δ-calculus that can be pictorially displayed in a "Δ-chair" (see Definition 9). Following the same style as in the Barendrengt et al. book [5], the edges in the chair represent an inclusion relation over the set of derivable judgments.

Section 3 shows a number of typable examples in the systems presented in the Δ-chair: each example is provided with a corresponding type assignment derivation of its essence. Some historical examples of Pottinger [39], Hindley [25] and Ben-Yelles [6] are essentially re-decorated and inhabited (when possible) in the Δ-calculus. The aim of this section is both to make the reader comfortable with the different intersection typed systems, and to give a first intuition of the correspondence between Church-style and Curry-style calculi.

Section 4 proves the metatheory for all the systems in the Δ-chair: Church-Rosser, unicity of type, subject reduction, strong normalization, decidability of type checking and type reconstruction and studies the relations between intersection type assignment systems à la Curry and the corresponding intersection typed systems à la Church. Notions of soundness, completeness and isomorphism will relate type assignment and typed systems. We also show how to remove type coercions Δ^τ defining a translation function, denoted by $\|_\|$, inspired by the one of Tannen et al. [8]: the intuition of the translation is that if Δ has type σ and $\sigma \leqslant_{\mathcal{T}} \tau$, then $\|\sigma \leqslant_{\mathcal{T}} \tau\|$ is a Δ-term of type $\sigma \to \tau$, $(\|\sigma \leqslant_{\mathcal{T}} \tau\|\; \|\Delta\|)$ has type τ and $\wr \|\sigma \leqslant_{\mathcal{T}} \tau\| \wr$ is the identity $\lambda x.x$.

1.1 λ-calculi with intersection types à la Church

Several calculi à la Church appeared in the literature: they capture the power of intersection types; we briefly review them.

The Forsythe programming language by Reynolds [41] annotates a λ-abstraction with types as in $\lambda x{:}\sigma_1|\cdots|\sigma_n.M$. However, there is no typed term whose type erasure is the combinator $\mathsf{K} \equiv \lambda x.\lambda y.x$, with the type $(\sigma \to \sigma \to \sigma) \cap (\tau \to \tau \to \tau)$.

Pierce [37] improves Forsythe by using a **for** construct to build ad hoc polymorphic typing, as in $\textbf{for}\, \alpha \in \{\sigma, \tau\}.\lambda x{:}\alpha.\lambda y{:}\alpha.x$. However, there is no typed term whose type erasure is $\lambda x.\lambda y.\lambda z.(x\, y\, , x\, z)$, with the type
$((\sigma \to \rho) \cap (\tau \to \rho') \to \sigma \to \tau \to \rho \times \rho') \cap ((\sigma \to \sigma) \cap (\sigma \to \sigma) \to \sigma \to \sigma \to \sigma \times \sigma)$.

Freeman and Pfenning [21] introduced refinement types, that is types that allow *ad hoc* polymorphism for ML constructors. Intuitively, refinement types can be seen as subtypes of a standard type: the user first defines a type and then the refinement types of this type. The main motivation for these refinement types is to allow non-exhaustive pattern matching, which becomes exhaustive for a given refinement of the type of the argument. As an example, we can define a type `boolexp` for boolean expressions, with constructors `True`, `And`, `Not` and `Var`, and a refinement type `ground` for boolean expressions without variables, with the same constructors except `Var`: then, the constructor `True` has type $\texttt{boolexp} \cap \texttt{ground}$, the constructor `And` has type $(\texttt{boolexp} * \texttt{boolexp} \to \texttt{boolexp}) \cap (\texttt{ground} * \texttt{ground} \to \texttt{ground})$ and so on. However, intersection is meaningful only when using constructors.

Wells et al. [49] introduced λ^{CIL}, a typed intermediate λ-calculus for optimizing compilers for higher-order programming languages. The calculus features intersection, union and flow types, the latter being useful to optimize data representation. λ^{CIL} can faithfully encode an intersection type assignment derivation by introducing the concept of virtual tuple, i.e. a special kind of pair whose type erasure leads to exactly the same untyped λ-term. A parallel context and parallel substitution, similar to the notion of [29, 30], is defined to reduce expressions in parallel inside a virtual tuple. Subtyping is defined only on flow types and not on intersection types: this system can encode the $\lambda_\cap^{\mathrm{CD}}$ type assignment system.

Wells and Haak [50] introduced λ^{B}, a more compact typed calculus encoding of λ^{CIL}: in fact, by comparing Fig. 1 and Fig. 2 of [50] we can see that the set of typable terms with intersection types of λ^{CIL} and λ^{B} are the same. In that paper, virtual tuples are removed by introducing branching terms, typable with branching types, the latter representing intersection type schemes. Two operations on types and terms are defined, namely expand, expanding the branching shape of type annotations when a term is substituted into a new context, and select, to choose the correct branch in terms and types. As there are no virtual tuples, reductions do not need to be done in parallel. As in [49], the $\lambda_\cap^{\mathrm{CD}}$ type assignment system can be encoded.

Frisch et al. [22] designed a typed system with intersection, union, negation and recursive types. The authors inherit the usual problem of having a domain space $\mathcal{D}$ that contains all the terms and, at the same time, all the functions from $\mathcal{D}$ to $\mathcal{D}$. They prevent this by having an auxiliary domain space which is the disjoint union of $\mathcal{D}^2$ and $\mathcal{P}(\mathcal{D}^2)$. The authors interpret types as sets in a well-suited model where the set-inspired type constructs are interpreted as the corresponding to set-theoretical constructs. Moreover, the model manages higher-order functions in an elegant way. The subtyping relation is defined as a relation on the set-theoretical interpretation $[\![_]\!]$ of the types. For instance, the problem $\sigma \cap \tau \leqslant \sigma$ will be interpreted as $[\![\sigma]\!] \cap [\![\tau]\!] \subseteq [\![\sigma]\!]$, where $\cap$ becomes the set intersection operator, and the decision program actually decides whether $([\![\sigma]\!] \cap [\![\tau]\!]) \cap \overline{[\![\sigma]\!]}$ is the empty set.

Bono et al. [7] introduced a relevant and strict parallel term constructor to build inhabitants of intersections and a simple call-by-value parallel reduction strategy. In that paper, an infinite number of constants $c^{\sigma \Rightarrow \tau}$ is applied to typed variables x^σ such that $c^{\sigma \Rightarrow \tau}\, x^\sigma$ is upcasted to type τ. The paper also uses a local renaming typing rule, which changes the type decoration in λ-abstractions, as well as coercions. Term synchronicity in the tuples is guaranteed by the typing rules. The calculus uses van Bakel's strict version [46] of the $\mathcal{T}_{\mathrm{CD}}$ intersection type theory.

1.2 Logics for intersection types

Proof-functional (or strong) logical connectives, introduced by Pottinger [39], take into account the shape of logical proofs, thus allowing for polymorphic features of proofs to be made explicit in formulæ. This differs from classical or intuitionistic connectives where the meaning of a compound formula is only dependent on the truth value or the provability of its subformulæ.

Pottinger was the first to consider the intersection $\cap$ as a proof-functional connective. He contrasted it to the intuitionistic connective $\wedge$ as follows: *"The intuitive meaning of $\cap$ can be explained by saying that to assert $A \cap B$ is to assert that one has a reason for asserting A which is also a reason for asserting B, while to assert $A \wedge B$ is to assert that one has a pair of reasons, the first of which is a reason for asserting A and the second of which is a reason for asserting B"*.

A simple example of a logical theorem involving intuitionistic conjunction which does not hold for proof-functional conjunction is $(A \supset A) \wedge (A \supset B \supset A)$. Otherwise there would exist a term which behaves both as I and as K. Later, Lopez-Escobar [32] and Mints [33] investigated extensively logics featuring both proof-functional and intuitionistic connectives especially in the context of realizability interpretations.

It is not immediate to extend the judgments-as-types Curry-Howard paradigm to logics supporting proof-functional connectives. These connectives need to compare the shapes of derivations and do not just take into account their provability, i.e. the inhabitation of the corresponding type.

There are many proposals to find a suitable logics to fit intersection types; among them we cite [48, 42, 34, 10, 7, 38], and previous papers by the authors [16, 31, 43].

1.3 Raising the Δ-calculus to a Δ-framework.

Our long term goal is to build a prototype of a theorem prover based on the Δ-calculus and proof-functional logic. Recently [27], we have extended a subset of the generic Δ-calculus with other proof-functional operators like union types, relevant arrow types, together with dependent types as in the Edinburgh Logical Framework [24]: a preliminary implementation of a type checker appeared in [43] by the authors. In a nutshell:

Strong disjunction is a proof-functional connective that can be interpreted as the union type $\cup$ [16, 43]: it contrasts with the intuitionistic connective $\vee$. As Pottinger did for intersection, we could informally say that asserting $(A \cup B) \supset C$ corresponds to have a same reason for both $A \supset C$ and $B \supset C$.

A simple example of a logical theorem involving intuitionistic disjunction which does not hold for strong disjunction is $((A \supset B) \cup B) \supset A \supset B$. Otherwise there would exist a term which behaves both as I and as K.

Minimal type theory $\leqslant_{\min}$

(refl)	$\sigma \leqslant \sigma$	(incl)	$\sigma \cap \tau \leqslant \sigma, \sigma \cap \tau \leqslant \tau$
(glb)	$\rho \leqslant \sigma, \rho \leqslant \tau \Rightarrow \rho \leqslant \sigma \cap \tau$	(trans)	$\sigma \leqslant \tau, \tau \leqslant \rho \Rightarrow \sigma \leqslant \rho$

Axiom schemes

$(\mathtt{U}_{top})$	$\sigma \leqslant \mathtt{U}$	$(\mathtt{U}_{\rightarrow})$	$\mathtt{U} \leqslant \sigma \rightarrow \mathtt{U}$
$(\rightarrow\cap)$	$(\sigma \rightarrow \tau) \cap (\sigma \rightarrow \rho) \leqslant \sigma \rightarrow (\tau \cap \rho)$		

Rule scheme

$(\rightarrow)$ $\quad \sigma_2 \leqslant \sigma_1, \tau_1 \leqslant \tau_2 \Rightarrow \sigma_1 \rightarrow \tau_1 \leqslant \sigma_2 \rightarrow \tau_2$

Figure 1 Minimal type theory $\leqslant_{\min}$, axioms and rule schemes (see Fig. 13.2 and 13.3 of [5]).

$$\frac{x{:}\sigma \in B}{B \vdash_\cap^{\mathcal{T}} x : \sigma}\ (ax) \qquad \frac{B, x{:}\sigma \vdash_\cap^{\mathcal{T}} M : \tau}{B \vdash_\cap^{\mathcal{T}} \lambda x.M : \sigma \rightarrow \tau}\ (\rightarrow I)$$

$$\frac{B \vdash_\cap^{\mathcal{T}} M : \sigma \quad B \vdash_\cap^{\mathcal{T}} M : \tau}{B \vdash_\cap^{\mathcal{T}} M : \sigma \cap \tau}\ (\cap I) \qquad \frac{B \vdash_\cap^{\mathcal{T}} M : \sigma \rightarrow \tau \quad B \vdash_\cap^{\mathcal{T}} N : \sigma}{B \vdash_\cap^{\mathcal{T}} M\, N : \tau}\ (\rightarrow E)$$

$$\frac{B \vdash_\cap^{\mathcal{T}} M : \sigma \cap \tau}{B \vdash_\cap^{\mathcal{T}} M : \sigma}\ (\cap E_1) \qquad \frac{B \vdash_\cap^{\mathcal{T}} M : \sigma \cap \tau}{B \vdash_\cap^{\mathcal{T}} M : \tau}\ (\cap E_2)$$

$$\frac{\mathtt{U} \in \mathbb{A}}{B \vdash_\cap^{\mathcal{T}} M : \mathtt{U}}\ (top) \qquad \frac{B \vdash_\cap^{\mathcal{T}} M : \sigma \quad \sigma \leqslant_{\mathcal{T}} \tau}{B \vdash_\cap^{\mathcal{T}} M : \tau}\ (\leqslant_{\mathcal{T}})$$

Figure 2 Generic intersection type assignment system $\lambda_\cap^{\mathcal{T}}$ (see Figure 13.8 of [5]). Although rules $(\cap E_i)$ are derivable with $\leqslant_{\min}$, we add them for clarity.

Strong (relevant) implication is yet another proof-functional connective that was interpreted in [2] as a relevant arrow type $\rightarrow_r$. As explained in [2], it can be viewed as a special case of implication whose related function space is the simplest one, namely the one containing only the identity function. Because the operators $\supset$ and $\rightarrow_r$ differ, $A \rightarrow_r B \rightarrow_r A$ is not derivable.

Dependent types, as introduced in the Edinburgh Logical Framework [24] by Harper et al., allows considering proofs as first-class citizens albeit differently with respect to proof-functional logics. The interaction of both dependent and proof-functional operators is intriguing: the former mentions proofs explicitly, while the latter mentions proofs implicitly. Their combination therefore opens up new possibilities of formal reasoning on proof-theoretic semantics.

2 Syntax, Reduction and Types

▶ **Definition 1** (Type atoms, type syntax, type theories and type assignment systems). *We briefly review some basic definition from Subsection 13.1 of [5], in order to define type assignment systems. The set of atoms, intersection types, intersection type theories and intersection type assignment systems are defined as follows:*

1. **(Atoms).** *Let $\mathbb{A}$ be a set of symbols which we will call type atoms, and let $\mathtt{U}$ be a special type atom denoting the universal type. In particular, we will use $\mathbb{A}_\infty = \{\mathtt{a}_i \mid i \in \mathbb{N}\}$ with $\mathtt{a}_i$ being different from $\mathtt{U}$ and $\mathbb{A}_\infty^{\mathtt{U}} = \mathbb{A}_\infty \cup \{\mathtt{U}\}$.*
2. **(Syntax).** *The syntax of intersection types, parametrized by $\mathbb{A}$, is: $\sigma ::= \mathbb{A} \mid \sigma \rightarrow \sigma \mid \sigma \cap \sigma$.*

Table 1 Type theories $\lambda_\cap^{\mathrm{CD}}$, $\lambda_\cap^{\mathrm{CDS}}$, $\lambda_\cap^{\mathrm{CDV}}$, and $\lambda_\cap^{\mathrm{BCD}}$. The ref. column refers to the original article these theories come from.

$\lambda_\cap^{\mathcal{T}}$	$\mathcal{T}$	$\mathbb{A}$	$\leqslant_{\min}$ plus	ref.
$\lambda_\cap^{\mathrm{CD}}$	$\mathcal{T}_{\mathrm{CD}}$	$\mathbb{A}_\infty$	–	[12]
$\lambda_\cap^{\mathrm{CDS}}$	$\mathcal{T}_{\mathrm{CDS}}$	$\mathbb{A}_\infty^{\mathtt{U}}$	$(\mathtt{U}_{top})$	[13]
$\lambda_\cap^{\mathrm{CDV}}$	$\mathcal{T}_{\mathrm{CDV}}$	$\mathbb{A}_\infty$	$(\to), (\to\cap)$	[14]
$\lambda_\cap^{\mathrm{BCD}}$	$\mathcal{T}_{\mathrm{BCD}}$	$\mathbb{A}_\infty^{\mathtt{U}}$	$(\to), (\to\cap), (\mathtt{U}_{top}), (\mathtt{U}\to)$	[4]

3. **(Intersection type theories $\mathcal{T}$).** *An intersection type theory $\mathcal{T}$ is a set of sentences of the form $\sigma \leqslant \tau$ satisfying at least the axioms and rules of the minimal type theory $\leqslant_{min}$ defined in Figure 1. The type theories $\mathcal{T}_{\mathrm{CD}}, \mathcal{T}_{\mathrm{CDV}}, \mathcal{T}_{\mathrm{CDS}}$, and $\mathcal{T}_{\mathrm{BCD}}$ are the smallest type theories over $\mathbb{A}$ satisfying the axioms and rules given in Table 1. We write $\mathcal{T}_1 \sqsubseteq \mathcal{T}_2$ if, for all σ, τ such that $\sigma \leqslant_{\mathcal{T}_1} \tau$, we have that $\sigma \leqslant_{\mathcal{T}_2} \tau$. In particular, $\mathcal{T}_{\mathrm{CD}} \sqsubseteq \mathcal{T}_{\mathrm{CDV}} \sqsubseteq \mathcal{T}_{\mathrm{BCD}}$ and $\mathcal{T}_{\mathrm{CD}} \sqsubseteq \mathcal{T}_{\mathrm{CDS}} \sqsubseteq \mathcal{T}_{\mathrm{BCD}}$. We will sometimes note, for instance,* BCD *instead of $\mathcal{T}_{\mathrm{BCD}}$.*
4. **(Intersection type assignment systems $\lambda_\cap^{\mathcal{T}}$).** *We define in Figure 2 an infinite collection of type assignment systems parametrized by a set of atoms $\mathbb{A}$ and a type theory $\mathcal{T}$. We name four particular type assignment systems in the table below, which is an excerpt from Figure 13.4 of [5]. $B \vdash_\cap^{\mathcal{T}} M : \sigma$ denotes a derivable type assignment judgment in the type assignment system $\lambda_\cap^{\mathcal{T}}$. Type checking is not decidable for $\lambda_\cap^{\mathrm{CD}}$, $\lambda_\cap^{\mathrm{CDV}}$, $\lambda_\cap^{\mathrm{CDS}}$, and $\lambda_\cap^{\mathrm{BCD}}$.*

2.1 The Δ-calculi

Intersection type assignment systems and Δ-calculi have in common their type syntax and intersection type theories. The generic syntax of the Δ-calculus is defined as follows.

▶ **Definition 2** (Generic Δ-calculus syntax).

$$\Delta \quad ::= \quad u_\Delta \mid x \mid \lambda x{:}\sigma.\Delta \mid \Delta\,\Delta \mid \langle \Delta\,,\Delta \rangle \mid pr_i\,\Delta \mid \Delta^\sigma \qquad\qquad i \in \{1,2\}$$

Intuitively, u_Δ ranges over an infinite set of constants, indexed with a particular Δ-term. Δ^σ denotes an explicit coercion of Δ to type σ. The expression $\langle \Delta\,,\Delta \rangle$ denotes a pair that, following the Lopez-Escobar jargon [32], we call "strong pair" with respective projections pr_1 and pr_2. Note that pr_i is not a term: if it were a term, then pr_i should have several types, or a parametric polymorphic type. But our calculi do not have parametric polymorphism, and they have unicity of typing.

The essence function $\wr_\wr$ is an erasing function mapping typed Δ-terms into pure λ-terms. It is defined as follows.

▶ **Definition 3** (Essence function).

$$\wr x \wr \stackrel{def}{=} x \qquad \wr \Delta^\sigma \wr \stackrel{def}{=} \wr \Delta \wr \qquad \wr u_\Delta \wr \stackrel{def}{=} \wr \Delta \wr$$

$$\wr \lambda x{:}\sigma.\Delta \wr \stackrel{def}{=} \lambda x.\wr \Delta \wr \qquad \wr \Delta_1\,\Delta_2 \wr \stackrel{def}{=} \wr \Delta_1 \wr \wr \Delta_2 \wr$$

$$\wr \langle \Delta_1\,,\Delta_2 \rangle \wr \stackrel{def}{=} \wr \Delta_1 \wr \qquad \wr pr_i\,\Delta \wr \stackrel{def}{=} \wr \Delta \wr \quad i \in \{1,2\}$$

One could argue that the choice of $\wr \langle \Delta_1\,,\Delta_2 \rangle \wr \stackrel{def}{=} \wr \Delta_1 \wr$ is arbitrary and could have been replaced with $\wr \langle \Delta_1\,,\Delta_2 \rangle \wr \stackrel{def}{=} \wr \Delta_2 \wr$. However, the typing rules will ensure that, if $\langle \Delta_1\,,\Delta_2 \rangle$

is typable, then, for some suitable equivalence relation $\mathcal{R}$, we have that $\wr\Delta_1\wr\,\mathcal{R}\,\wr\Delta_2\wr$. Thus, strong pairs can be viewed as constrained cartesian products.

The generic reduction semantics reduces terms of the Δ-calculus as follows.

▶ **Definition 4** (Generic reduction semantics). *Syntactical equality is denoted by $\equiv$.*

1. **(Substitution).** *Substitution on Δ-terms is defined as usual, with the additional rules:*

$$u_{\Delta_1}[\Delta_2/x] \stackrel{def}{=} u_{(\Delta_1[\Delta_2/x])} \qquad and \qquad \Delta_1^{\sigma}[\Delta_2/x] \stackrel{def}{=} (\Delta_1[\Delta_2/x])^{\sigma}$$

2. **(One-step reduction).** *We define two notions of reduction:*

$$\begin{array}{llll} (\lambda x{:}\sigma.\Delta_1)\,\Delta_2 & \longrightarrow_{\beta} & \Delta_1[\Delta_2/x] & (\beta) \\ pr_i\,\langle\Delta_1\,,\Delta_2\rangle & \longrightarrow_{pr_i} & \Delta_i \quad i\in\{1,2\} & (pr_i) \end{array}$$

Observe that $(\lambda x{:}\sigma.\Delta_1)^{\sigma}\,\Delta_2$ is not a redex, because the λ-abstraction is coerced. The contextual closure is defined as usual except for reductions inside the index of u_Δ that are forbidden (even though substitutions are propagated). We write $\longrightarrow_{\beta pr_i}$ for the contextual closure of the (β) and (pr_i) notions of reduction. We also define a synchronous contextual closure, which is like the usual contextual closure except for the strong pairs, as defined in point (3). Synchronous contextual closure of the notions of reduction generates the reduction relation $\longrightarrow^{\parallel}_{\beta pr_i}$.

3. **(Synchronous closure of $\longrightarrow^{\parallel}$).** *Synchronous closure is defined on the strong pairs with the following constraint:*

$$\frac{\Delta_1 \longrightarrow^{\parallel} \Delta_1' \quad \Delta_2 \longrightarrow^{\parallel} \Delta_2' \quad \wr\Delta_1'\wr \equiv \wr\Delta_2'\wr}{\langle\Delta_1\,,\Delta_2\rangle \longrightarrow^{\parallel} \langle\Delta_1'\,,\Delta_2'\rangle}\ (Clos^{\parallel})$$

Note that we reduce in the two components of the strong pair;

4. **(Multistep reduction).** *We write $\twoheadrightarrow_{\beta pr_i}$ (resp. $\twoheadrightarrow^{\parallel}_{\beta pr_i}$) as the reflexive and transitive closure of $\longrightarrow_{\beta pr_i}$ (resp. $\longrightarrow^{\parallel}_{\beta pr_i}$);*
5. **(Congruence).** *We write $=_{\beta pr_i}$ as the congruence relation generated by $\twoheadrightarrow_{\beta pr_i}$.*

We mostly consider βpr_i-reductions, thus to ease the notation we omit the subscript in βpr_i-reductions.

Note that η-reduction can be defined as in the untyped λ-calculus. The next definition introduces a notion of synchronization inside strong pairs. This notion is also valid in untyped calculi.

▶ **Definition 5** (Synchronization). *A Δ-term is synchronous if and only if, for all its subterms of the shape $\langle\Delta_1\,,\Delta_2\rangle$, we have that $\wr\Delta_1\wr \equiv \wr\Delta_2\wr$.*

It is easy to verify that $\longrightarrow^{\parallel}$ preserves synchronization, while this is not the case for $\longrightarrow$. The next definition introduces a generic intersection typed system for the Δ-calculus that is parametrizable by suitable equivalence relations on pure λ-terms $\mathcal{R}$ and type theories $\mathcal{T}$.

▶ **Definition 6** (Generic intersection typed system). *The generic intersection typed system is defined in Figure 3. We denote by $\Delta^{\mathcal{T}}_{\mathcal{R}}$ a particular typed system with the type theory $\mathcal{T}$ and under an equivalence relation $\mathcal{R}$ and by $B \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta : \sigma$ a corresponding typing judgment.*

The typing rules are intuitive for a calculus à la Church except rules $(\cap I)$, (top) and $(\leqslant_{\mathcal{T}})$. The typing rule for a strong pair $(\cap I)$ is similar to the typing rule for a cartesian product, except for the side-condition $\wr\Delta_1\wr\,\mathcal{R}\,\wr\Delta_2\wr$, forcing the two parts of the strong pair to have essences compatible under $\mathcal{R}$, thus making a strong pair a special case of a cartesian pair.

$$\frac{\mathtt{U} \in \mathbb{A}}{B \vdash^{\mathcal{T}}_{\mathcal{R}} u_\Delta : \mathtt{U}}\ (top) \qquad \frac{x{:}\sigma \in B}{B \vdash^{\mathcal{T}}_{\mathcal{R}} x : \sigma}\ (ax) \qquad \frac{B, x{:}\sigma \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta : \tau}{B \vdash^{\mathcal{T}}_{\mathcal{R}} \lambda x{:}\sigma.\Delta : \sigma \to \tau}\ (\to I)$$

$$\frac{B \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta_1 : \sigma \quad B \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta_2 : \tau \quad \wr \Delta_1 \wr \mathcal{R} \wr \Delta_2 \wr}{B \vdash^{\mathcal{T}}_{\mathcal{R}} \langle \Delta_1 , \Delta_2 \rangle : \sigma \cap \tau}\ (\cap I) \qquad \frac{B \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta_1 : \sigma \to \tau \quad B \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta_2 : \sigma}{B \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta_1\, \Delta_2 : \tau}\ (\to E)$$

$$\frac{B \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta : \sigma \cap \tau}{B \vdash^{\mathcal{T}}_{\mathcal{R}} pr_1\, \Delta : \sigma}\ (\cap E_1) \qquad \frac{B \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta : \sigma \cap \tau}{B \vdash^{\mathcal{T}}_{\mathcal{R}} pr_2\, \Delta : \tau}\ (\cap E_2) \qquad \frac{B \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta : \sigma \quad \sigma \leqslant_{\mathcal{T}} \tau}{B \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta^{\tau} : \tau}\ (\leqslant_{\mathcal{T}})$$

Figure 3 Generic intersection typed system $\Delta^{\mathcal{T}}_{\mathcal{R}}$.

For instance, $\langle \lambda x{:}\sigma.\lambda y{:}\tau.x , \lambda x{:}\sigma.x \rangle$ is not typable in $\Delta^{\mathcal{T}}_{\equiv}$; $\langle (\lambda x{:}\sigma.x)\, y , y \rangle$ is not typable in $\Delta^{\mathcal{T}}_{\equiv}$ but it is in $\Delta^{\mathcal{T}}_{=_\beta}$; $\langle x , \lambda y{:}\sigma.((\lambda z{:}\tau.z)\, x)\, y \rangle$ is not typable in $\Delta^{\mathcal{T}}_{\equiv}$ nor $\Delta^{\mathcal{T}}_{=_\beta}$ but it is in $\Delta^{\mathcal{T}}_{=_{\beta\eta}}$. In the typing rule (top), the subscript Δ in u_Δ is not necessarily typable so $\wr u_\Delta \wr$ can easily be any arbitrary λ-term. The typing rule $(\leqslant_{\mathcal{T}})$ allows to change the type of a Δ-term from σ to τ if $\sigma \leqslant_{\mathcal{T}} \tau$: the term in the conclusion must record this change with an explicit type coercion $_^{\tau}$, producing the new term Δ^{τ}: explicit type coercions are important to keep the unicity of typing derivations.

The next definition introduces a partial order over equivalence relations on pure λ-terms and an inclusion over typed systems as follows.

▶ **Definition 7** ($\mathcal{R}$ and $\sqsubseteq$).

1. *Let $\mathcal{R} \in \{\equiv, =_\beta, =_{\beta\eta}\}$. $\mathcal{R}_1 \sqsubseteq \mathcal{R}_2$ if, for all pure λ-terms M, N such that $M\ \mathcal{R}_1\ N$, we have that $M\ \mathcal{R}_2\ N$;*
2. *$\Delta^{\mathcal{T}_1}_{\mathcal{R}_1} \sqsubseteq \Delta^{\mathcal{T}_2}_{\mathcal{R}_2}$ if, whenever $B \vdash^{\mathcal{T}_1}_{\mathcal{R}_1} \Delta : \sigma$, we have $B \vdash^{\mathcal{T}_2}_{\mathcal{R}_2} \Delta : \sigma$.*

▶ **Lemma 8.**

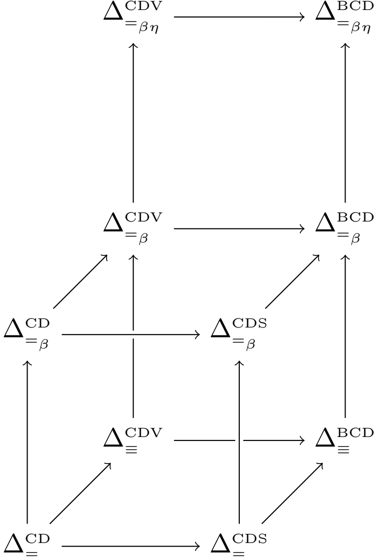
1. *$\Delta^{\mathrm{CD}}_{\mathcal{R}} \sqsubseteq \Delta^{\mathrm{CDS}}_{\mathcal{R}} \sqsubseteq \Delta^{\mathrm{BCD}}_{\mathcal{R}}$ and $\Delta^{\mathrm{CD}}_{\mathcal{R}} \sqsubseteq \Delta^{\mathrm{CDV}}_{\mathcal{R}} \sqsubseteq \Delta^{\mathrm{BCD}}_{\mathcal{R}}$;*
2. *$\Delta^{\mathcal{T}_1}_{\mathcal{R}_1} \sqsubseteq \Delta^{\mathcal{T}_2}_{\mathcal{R}_2}$ if $\mathcal{T}_1 \sqsubseteq \mathcal{T}_2$ and $\mathcal{R}_1 \sqsubseteq \mathcal{R}_2$.*

2.2 The Δ-chair

The next definition classifies ten typed systems for the Δ-calculus: some of them already appeared (sometime with a different notation) in the literature by the present authors.

▶ **Definition 9** (Δ-chair). *Ten typed systems $\Delta^{\mathcal{T}}_{\mathcal{R}}$ can be drawn pictorially in a Δ-chair, where the arrows represent an inclusion relation. $\Delta^{\mathrm{CD}}_{\equiv}$ corresponds roughly to [29, 30] (in the expression $M@\Delta$, M is the essence of Δ) and in its intersection part to [43]; $\Delta^{\mathrm{CDS}}_{\equiv}$ corresponds roughly in its intersection part to [17], $\Delta^{\mathrm{BCD}}_{\equiv}$ corresponds in its intersection part to [31], $\Delta^{\mathrm{CD}}_{=_{\beta\eta}}$ corresponds in its intersection part to [16]. The other typed systems are basically new. The main properties of these systems are:*

1. *All the $\Delta^{\mathcal{T}}_{\equiv}$ systems enjoys the synchronous subject reduction property, the other systems also enjoy ordinary subject reduction (Theorem 29);*
2. *All the systems strongly normalize (Theorem 33);*
3. *All the systems correspond to the to original type assignment systems except $\Delta^{\mathrm{CD}}_{=_\beta}$, $\Delta^{\mathrm{CDV}}_{=_\beta}$, $\Delta^{\mathrm{CDV}}_{=_{\beta\eta}}$ and $\Delta^{\mathrm{BCD}}_{=_{\beta\eta}}$ (Theorem 35);*
4. *Type checking and type reconstruction are decidable for all the systems, except $\Delta^{\mathrm{CDS}}_{=_\beta}$, $\Delta^{\mathrm{BCD}}_{=_\beta}$, and $\Delta^{\mathrm{BCD}}_{=_{\beta\eta}}$ (Theorem 37).*



3 Examples

This section shows examples of typed derivations $\Delta^{\mathcal{T}}_{\mathcal{R}}$ and highlights the corresponding type assignment judgment in $\lambda^{\mathcal{T}}_{\cap}$ they correspond to, in the sense that we have a derivation $B \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta : \sigma$ and another derivation $B \vdash^{\mathcal{T}}_{\cap} \wr \Delta \wr : \sigma$. The correspondence between intersection typed systems $\Delta^{\mathcal{T}}_{\mathcal{R}}$ and intersection type assignment $\lambda^{\mathcal{T}}_{\cap}$ will be defined in Subsection 5.1.

▶ **Example 10** (Polymorphic identity). In all of the intersection type assignment systems $\lambda^{\mathcal{T}}_{\cap}$ we can derive $\vdash^{\mathcal{T}}_{\cap} \lambda x.x : (\sigma \to \sigma) \cap (\tau \to \tau)$. A corresponding Δ-term is: $\langle \lambda x{:}\sigma.x \,, \lambda x{:}\tau.x \rangle$ that can be typed in all of the typed systems of the Δ-chair as follows

$$\dfrac{\dfrac{x{:}\sigma \vdash^{\mathcal{T}}_{\mathcal{R}} x : \sigma}{\vdash^{\mathcal{T}}_{\mathcal{R}} \lambda x{:}\sigma.x : \sigma \to \sigma} \quad \dfrac{x{:}\tau \vdash^{\mathcal{T}}_{\mathcal{R}} x : \tau}{\vdash^{\mathcal{T}}_{\mathcal{R}} \lambda x{:}\tau.x : \tau \to \tau} \quad \lambda x.x \;\mathcal{R}\; \lambda x.x}{\vdash^{\mathcal{T}}_{\mathcal{R}} \langle \lambda x{:}\sigma.x \,, \lambda x{:}\tau.x \rangle : (\sigma \to \sigma) \cap (\tau \to \tau)}$$

▶ **Example 11** (Auto application). In all of the intersection type assignment systems we can derive $\vdash^{\mathcal{T}}_{\cap} \lambda x.x\,x : ((\sigma \to \tau) \cap \sigma) \to \tau$. A corresponding Δ-term is: $\lambda x{:}(\sigma \to \tau) \cap \sigma.(pr_1\, x)(pr_2\, x)$ that can be typed in all of the typed systems of the Δ-chair as follows

$$\dfrac{\dfrac{\dfrac{x{:}(\sigma \to \tau) \cap \sigma \vdash^{\mathcal{T}}_{\mathcal{R}} x : (\sigma \to \tau) \cap \sigma}{x{:}(\sigma \to \tau) \cap \sigma \vdash^{\mathcal{T}}_{\mathcal{R}} pr_1\, x : \sigma \to \tau} \quad \dfrac{x{:}(\sigma \to \tau) \cap \sigma \vdash^{\mathcal{T}}_{\mathcal{R}} x : (\sigma \to \tau) \cap \sigma}{x{:}(\sigma \to \tau) \cap \sigma \vdash^{\mathcal{T}}_{\mathcal{R}} pr_2\, x : \sigma}}{x{:}(\sigma \to \tau) \cap \sigma \vdash^{\mathcal{T}}_{\mathcal{R}} (pr_1\, x)(pr_2\, x) : \tau}}{\vdash^{\mathcal{T}}_{\mathcal{R}} \lambda x{:}(\sigma \to \tau) \cap \sigma.(pr_1\, x)(pr_2\, x) : (\sigma \to \tau) \cap \sigma \to \tau}$$

▶ **Example 12** (Some examples in $\Delta^{\mathrm{CDS}}_{\mathcal{R}}$). In $\lambda^{\mathrm{CDS}}_{\cap}$ we can derive $\vdash^{\mathcal{T}_{\mathrm{CDS}}}_{\cap} (\lambda x.\lambda y.x) : \sigma \to \mathtt{U} \to \sigma$, and using this type assignment, we can derive $z{:}\sigma \vdash^{\mathcal{T}_{\mathrm{CDS}}}_{\cap} (\lambda x.\lambda y.x)\, z\, z : \sigma$. A corresponding

Δ-term is: $(\lambda x{:}\sigma.\lambda y{:}\mathtt{U}.x)\, z\, z^{\mathtt{U}}$ that can be typed in $\Delta^{\text{CDS}}_{\mathcal{R}}$ as follows

$$\dfrac{\dfrac{\dfrac{\dfrac{z{:}\sigma, x{:}\sigma, y{:}\mathtt{U} \vdash^{\mathcal{T}_{\text{CDS}}}_{\mathcal{R}} x : \sigma}{z{:}\sigma, x{:}\sigma \vdash^{\mathcal{T}_{\text{CDS}}}_{\mathcal{R}} \lambda y{:}\mathtt{U}.x : \mathtt{U} \to \sigma}}{z{:}\sigma \vdash^{\mathcal{T}_{\text{CDS}}}_{\mathcal{R}} \lambda x{:}\sigma.\lambda y{:}\mathtt{U}.x : \sigma \to \mathtt{U} \to \sigma} \quad z{:}\sigma \vdash^{\mathcal{T}_{\text{CDS}}}_{\mathcal{R}} z : \sigma}{z{:}\sigma \vdash^{\mathcal{T}_{\text{CDS}}}_{\mathcal{R}} (\lambda x{:}\sigma.\lambda y{:}\mathtt{U}.x)\, z : \mathtt{U} \to \sigma} \quad \dfrac{z{:}\sigma \vdash^{\mathcal{T}_{\text{CDS}}}_{\mathcal{R}} z : \sigma \quad \sigma \leqslant_{\mathcal{T}_{\text{CDS}}} \mathtt{U}}{z{:}\sigma \vdash^{\mathcal{T}_{\text{CDS}}}_{\mathcal{R}} z^{\mathtt{U}} : \mathtt{U}}}{z{:}\sigma \vdash^{\mathcal{T}_{\text{CDS}}}_{\mathcal{R}} (\lambda x{:}\sigma.\lambda y{:}\mathtt{U}.x)\, z\, z^{\mathtt{U}} : \sigma}$$

As another example, we can also derive $\vdash^{\mathcal{T}_{\text{CDS}}}_{\cap} \lambda x.x : \sigma \to \sigma \cap \mathtt{U}$. A corresponding Δ-term is: $\lambda x{:}\sigma.\langle x\,, x^{\mathtt{U}}\rangle$ that can be typed in $\Delta^{\text{CDS}}_{\mathcal{R}}$ as follows

$$\dfrac{\dfrac{x{:}\sigma \vdash^{\mathcal{T}_{\text{CDS}}}_{\mathcal{R}} x : \sigma \quad \dfrac{x{:}\sigma \vdash^{\mathcal{T}_{\text{CDS}}}_{\mathcal{R}} x : \sigma \quad \sigma \leqslant_{\mathcal{T}_{\text{CDS}}} \mathtt{U}}{x{:}\sigma \vdash^{\mathcal{T}_{\text{CDS}}}_{\mathcal{R}} x^{\mathtt{U}} : \mathtt{U}} \quad x \,\mathcal{R}\, x}{x{:}\sigma \vdash^{\mathcal{T}_{\text{CDS}}}_{\mathcal{R}} \langle x\,, x^{\mathtt{U}}\rangle : \sigma \cap \mathtt{U}}}{\vdash^{\mathcal{T}_{\text{CDS}}}_{\mathcal{R}} \lambda x{:}\sigma.\langle x\,, x^{\mathtt{U}}\rangle : \sigma \to \sigma \cap \mathtt{U}}$$

▶ **Example 13** (An example in $\Delta^{\text{CDV}}_{\mathcal{R}}$)**.** In $\lambda^{\text{CDV}}_{\cap}$ we can prove the commutativity of intersection, i.e. $\vdash^{\mathcal{T}_{\text{CDV}}}_{\cap} \lambda x.x : \sigma \cap \tau \to \tau \cap \sigma$. A corresponding Δ-term is:
$\langle \lambda x{:}\sigma \cap \tau.pr_2\, x\,, \lambda x{:}\sigma \cap \tau.pr_1\, x\rangle^{(\sigma\cap\tau)\to(\tau\cap\sigma)}$ that can be typed in $\Delta^{\text{CDV}}_{\mathcal{R}}$ as follows

$$\dfrac{\dfrac{\dfrac{\dfrac{x{:}\sigma \cap \tau \vdash^{\mathcal{T}_{\text{CDS}}}_{\mathcal{R}} x : \sigma \cap \tau}{x{:}\sigma \cap \tau \vdash^{\mathcal{T}_{\text{CDS}}}_{\mathcal{R}} pr_2\, x : \tau}}{\vdash^{\mathcal{T}_{\text{CDS}}}_{\mathcal{R}} \lambda x{:}\sigma \cap \tau.pr_2\, x : (\sigma \cap \tau) \to \tau} \quad \dfrac{\dfrac{x{:}\sigma \cap \tau \vdash^{\mathcal{T}_{\text{CDS}}}_{\mathcal{R}} x : \sigma \cap \tau}{x{:}\sigma \cap \tau \vdash^{\mathcal{T}_{\text{CDS}}}_{\mathcal{R}} pr_1\, x : \sigma}}{\vdash^{\mathcal{T}_{\text{CDS}}}_{\mathcal{R}} \lambda x{:}\sigma \cap \tau.pr_1\, x : (\sigma \cap \tau) \to \sigma} \quad \lambda x.x \,\mathcal{R}\, \lambda x.x}{\vdash^{\mathcal{T}_{\text{CDS}}}_{\mathcal{R}} \langle \lambda x{:}\sigma \cap \tau.pr_2\, x\,, \lambda x{:}\sigma \cap \tau.pr_1\, x\rangle : ((\sigma \cap \tau) \to \tau) \cap ((\sigma \cap \tau) \to \sigma)} \quad *}{\vdash^{\mathcal{T}_{\text{CDS}}}_{\mathcal{R}} \langle \lambda x{:}\sigma \cap \tau.pr_2\, x\,, \lambda x{:}\sigma \cap \tau.pr_1\, x\rangle^{(\sigma\cap\tau)\to(\tau\cap\sigma)} : (\sigma \cap \tau) \to (\tau \cap \sigma)}$$

where $*$ is $((\sigma \cap \tau) \to \tau) \cap ((\sigma \cap \tau) \to \sigma) \leqslant_{\mathcal{T}_{\text{CDV}}} (\sigma \cap \tau) \to (\tau \cap \sigma)$.

▶ **Example 14** (Another polymorphic identity in $\Delta^{\mathcal{T}}_{=_\beta}$)**.** In all the $\Delta^{\mathcal{T}}_{=_\beta}$ you can type the following Δ-term: $\langle \lambda x{:}\sigma.x\,, (\lambda x{:}\tau{\to}\tau.x)\,(\lambda x{:}\tau.x)\rangle$. The typing derivation is thus

$$\dfrac{\dfrac{x{:}\sigma \vdash^{\mathcal{T}}_{=_\beta} x : \sigma}{\vdash^{\mathcal{T}}_{=_\beta} \lambda x{:}\sigma.x : \sigma \to \sigma} \quad \dfrac{\dfrac{x{:}\tau \to \tau \vdash^{\mathcal{T}}_{=_\beta} x : \tau \to \tau}{\vdash^{\mathcal{T}}_{=_\beta} \lambda x{:}\tau{\to}\tau.x : (\tau \to \tau) \to (\tau \to \tau)} \quad \dfrac{x{:}\tau \vdash^{\mathcal{T}}_{=_\beta} x : \tau}{\vdash^{\mathcal{T}}_{=_\beta} \lambda x{:}\tau.x : \tau \to \tau}}{\vdash^{\mathcal{T}}_{=_\beta} (\lambda x{:}\tau{\to}\tau.x)\,(\lambda x{:}\tau.x) : \tau \to \tau} \quad \lambda x.x =_\beta (\lambda x.x)\,(\lambda x.x)}{\vdash^{\mathcal{T}}_{=_\beta} \langle \lambda x{:}\sigma.x\,, (\lambda x{:}\tau{\to}\tau.x)\,(\lambda x{:}\tau.x)\rangle : (\sigma \to \sigma) \cap (\tau \to \tau)}$$

▶ **Example 15** (Two examples in $\Delta^{\text{BCD}}_{\equiv}$ and $\Delta^{\text{BCD}}_{=_{\beta\eta}}$)**.** In $\lambda^{\text{BCD}}_{\cap}$ we can can type any term, including the non-terminating term $\Omega \stackrel{def}{=} (\lambda x.x\,x)\,(\lambda x.x\,x)$. More precisely, we have $\vdash^{\mathcal{T}_{\text{BCD}}}_{\cap} \Omega : \mathtt{U}$. A corresponding Δ-term whose essence is Ω is $(\lambda x{:}\mathtt{U}.x^{\mathtt{U}\to\mathtt{U}}\, x)\,(\lambda x{:}\mathtt{U}.x^{\mathtt{U}\to\mathtt{U}}\, x)^{\mathtt{U}}$ that can be typed in $\Delta^{\text{BCD}}_{\mathcal{R}}$ as follows

$$\dfrac{\dfrac{*}{\vdash^{\mathcal{T}_{\text{BCD}}}_{\mathcal{R}} \lambda x{:}\mathtt{U}.x^{\mathtt{U}\to\mathtt{U}}\, x : \mathtt{U} \to \mathtt{U}} \quad \dfrac{\dfrac{*}{\vdash^{\mathcal{T}_{\text{BCD}}}_{\mathcal{R}} \lambda x{:}\mathtt{U}.x^{\mathtt{U}\to\mathtt{U}}\, x : \mathtt{U} \to \mathtt{U}} \quad \mathtt{U} \to \mathtt{U} \leqslant_{\mathcal{T}_{\text{BCD}}} \mathtt{U}}{\vdash^{\mathcal{T}_{\text{BCD}}}_{\mathcal{R}} (\lambda x{:}\mathtt{U}.x^{\mathtt{U}\to\mathtt{U}}\, x)^{\mathtt{U}} : \mathtt{U}}}{\vdash^{\mathcal{T}_{\text{BCD}}}_{\mathcal{R}} (\lambda x{:}\mathtt{U}.x^{\mathtt{U}\to\mathtt{U}}\, x)\,(\lambda x{:}\mathtt{U}.x^{\mathtt{U}\to\mathtt{U}}\, x)^{\mathtt{U}} : \mathtt{U}}$$

where $*$ is

$$\dfrac{\dfrac{x{:}\mathtt{U} \vdash^{\mathcal{T}_{\text{BCD}}}_{\mathcal{R}} x : \mathtt{U} \quad \mathtt{U} \leqslant_{\mathcal{T}_{\text{BCD}}} \mathtt{U} \to \mathtt{U}}{x{:}\mathtt{U} \vdash^{\mathcal{T}_{\text{BCD}}}_{\mathcal{R}} x^{\mathtt{U}\to\mathtt{U}} : \mathtt{U} \to \mathtt{U}} \quad x{:}\mathtt{U} \vdash^{\mathcal{T}_{\text{BCD}}}_{\mathcal{R}} x : \mathtt{U}}{x{:}\mathtt{U} \vdash^{\mathcal{T}_{\text{BCD}}}_{\mathcal{R}} x^{\mathtt{U}\to\mathtt{U}}\, x : \mathtt{U}}$$

In $\lambda_\cap^{\text{BCD}}$ we can type $x{:}\mathtt{U} \to \mathtt{U} \vdash_\cap^{\mathcal{T}_{\text{BCD}}} x : (\mathtt{U} \to \mathtt{U}) \cap (\sigma \to \mathtt{U})$. A corresponding Δ-term whose essence is x is $\langle x \,, \lambda y{:}\sigma.x\, y^{\mathtt{U}} \rangle$ that can be typed in $\Delta_{=_{\beta\eta}}^{\text{BCD}}$ as follows

$$\dfrac{x{:}\mathtt{U} \to \mathtt{U} \vdash_{=_{\beta\eta}}^{\mathcal{T}_{\text{BCD}}} x : \mathtt{U} \to \mathtt{U} \qquad \dfrac{\dfrac{x{:}\mathtt{U} \to \mathtt{U}, y{:}\sigma \vdash_{=_{\beta\eta}}^{\mathcal{T}_{\text{BCD}}} x : \mathtt{U} \to \mathtt{U} \qquad \dfrac{x{:}\mathtt{U} \to \mathtt{U}, y{:}\sigma \vdash_{=_{\beta\eta}}^{\mathcal{T}_{\text{BCD}}} y : \sigma \quad \sigma \leqslant \mathtt{U}}{x{:}\mathtt{U} \to \mathtt{U}, y{:}\sigma \vdash_{=_{\beta\eta}}^{\mathcal{T}_{\text{BCD}}} y^{\mathtt{U}} : \mathtt{U}}}{x{:}\mathtt{U} \to \mathtt{U}, y{:}\sigma \vdash_{=_{\beta\eta}}^{\mathcal{T}_{\text{BCD}}} x\, y^{\mathtt{U}} : \mathtt{U}}}{x{:}\mathtt{U} \to \mathtt{U} \vdash_{=_{\beta\eta}}^{\mathcal{T}_{\text{BCD}}} \lambda y{:}\sigma.x\, y^{\mathtt{U}} : \sigma \to \mathtt{U}} \qquad x =_{\beta\eta} \lambda y.x\, y}{x{:}\mathtt{U} \to \mathtt{U} \vdash_{=_{\beta\eta}}^{\mathcal{T}_{\text{BCD}}} \langle x \,, \lambda y{:}\sigma.x\, y^{\mathtt{U}} \rangle : (\mathtt{U} \to \mathtt{U}) \cap (\sigma \to \mathtt{U})}$$

Note that the $=_{\beta\eta}$ condition has an interesting loophole, as it is well known that $\lambda_\cap^{\text{BCD}}$ does not enjoy $=_\eta$ conversion property. Theorem 35 will show that we can construct a Δ-term which does not correspond to any $\lambda_\cap^{\text{BCD}}$ derivation.

▶ **Example 16** (Pottinger)**.** The following examples can be typed in all the type theories of the Δ-chair (we also display the corresponding pure λ-terms typable in $\lambda_\cap^{\mathcal{T}}$). These are encodings from the examples à la Curry given by Pottinger in [39].

$\lambda x.\lambda y.x\, y$
$\vdash_{\mathcal{R}}^{\mathcal{T}} \lambda x{:}(\sigma \to \tau) \cap (\sigma \to \rho).\lambda y{:}\sigma.\langle (pr_1\, x)\, y) \,, (pr_2\, x)\, y \rangle : (\sigma \to \tau) \cap (\sigma \to \rho) \to \sigma \to \tau \cap \rho$

$\lambda x.\lambda y.x\, y$
$\vdash_{\mathcal{R}}^{\mathcal{T}} \lambda x{:}\sigma \to \tau \cap \rho.\langle \lambda y{:}\sigma.pr_1\, (x\, y) \,, \lambda y{:}\sigma.pr_2\, (x\, y) \rangle : (\sigma \to \tau \cap \rho) \to (\sigma \to \tau) \cap (\sigma \to \rho)$

$\lambda x.\lambda y.x\, y$
$\vdash_{\mathcal{R}}^{\mathcal{T}} \lambda x{:}\sigma \to \rho.\lambda y{:}\sigma \cap \tau.x\, (pr_1\, y) : (\sigma \to \rho) \to \sigma \cap \tau \to \rho$

$\lambda x.\lambda y.x$
$\vdash_{\mathcal{R}}^{\mathcal{T}} \lambda x{:}\sigma \cap \tau.\lambda y{:}\sigma.pr_2\, x : \sigma \cap \tau \to \sigma \to \tau$

$\lambda x.\lambda y.x\, y\, y$
$\vdash_{\mathcal{R}}^{\mathcal{T}} \lambda x{:}\sigma \to \tau \to \rho.\lambda y{:}\sigma \cap \tau.x\, (pr_1\, y)\, (pr_2\, y) : (\sigma \to \tau \to \rho) \to \sigma \cap \tau \to \rho$

$\lambda x.x$
$\vdash_{\mathcal{R}}^{\mathcal{T}} \lambda x{:}\sigma \cap \tau.pr_1\, x : \sigma \cap \tau \to \sigma$

$\lambda x.x$
$\vdash_{\mathcal{R}}^{\mathcal{T}} \lambda x{:}\sigma.\langle x \,, x \rangle : \sigma \to \sigma \cap \sigma$

$\lambda x.x$
$\vdash_{\mathcal{R}}^{\mathcal{T}} \lambda x{:}\sigma \cap (\tau \cap \rho).\langle \langle pr_1\, x \,, pr_1\, pr_2\, x \rangle \,, pr_2\, pr_2\, x \rangle : \sigma \cap (\tau \cap \rho) \to (\sigma \cap \tau) \cap \rho$

In the same paper, Pottinger lists some types that cannot be inhabited by any intersection type assignment ($\not\vdash_\cap^{\mathcal{T}}$) in an empty context, namely: $\sigma \to (\sigma \cap \tau)$, and $(\sigma \to \tau) \to (\sigma \to \rho) \to \sigma \to \tau \cap \rho$, and $((\sigma \cap \tau) \to \rho) \to \sigma \to \tau \to \rho$. It is not difficult to verify that the above types cannot be inhabited by any of the typed systems of the Δ-chair because of the failure of the essence condition in the strong pair type rule.

▶ **Example 17** (Intersection is not the conjunction operator)**.** This counter-example is from the corresponding counter-example à la Curry given by Hindley [26] and Ben-Yelles [6]. The intersection type $(\sigma \to \sigma) \cap ((\sigma \to \tau \to \rho) \to (\sigma \to \tau) \to \sigma \to \rho)$ where the left part of the intersection corresponds to the type for the combinator I and the right part for the combinator S cannot be assigned to a pure λ-term. Analogously, the same intersection type cannot be assigned to any Δ-term.

3.1 On synchronization and subject reduction

For the typed systems $\Delta^{\mathcal{T}}_{\equiv}$, strong pairs have an intrinsic notion of synchronization: some redexes need to be reduced in a synchronous fashion unless we want to create meaningless Δ-terms that cannot be typed. Consider the Δ-term $\langle (\lambda x{:}\sigma.x)\, y \,,(\lambda x{:}\sigma.x)\, y\rangle$: if we use the $\longrightarrow$ reduction relation, then the following reduction paths are legal:

$$\langle (\lambda x{:}\sigma.x)\, y \,,(\lambda x{:}\sigma.x)\, y\rangle \begin{array}{l} \nearrow^{\beta} \langle (\lambda x{:}\sigma.x)\, y \,, y\rangle \searrow_{\beta} \\ \searrow_{\beta} \langle y \,,(\lambda x{:}\sigma.x)\, y\rangle \nearrow^{\beta} \end{array} \langle y \,, y\rangle$$

More precisely, the first and second redexes are rewritten asynchronously, thus they cannot be typed in any typed system $\Delta^{\mathcal{T}}_{\equiv}$, because we fail to check the left and the right part of the strong pair to be the same: the $\longrightarrow^{\parallel}$ reduction relation prevents this loophole and allows to type all redexes. In summary, $\longrightarrow^{\parallel}$ can be thought of as the natural reduction relation for the typed systems $\Delta^{\mathcal{T}}_{\equiv}$.

4 Metatheory of $\Delta^{\mathcal{T}}_{\mathcal{R}}$

For lack of space all proofs are omitted: the interested reader can find more technical details in the related full version at `https://arxiv.org/abs/1803.09660`.

4.1 General properties

Unless specified, all properties applies to the intersection typed systems $\Delta^{\mathcal{T}}_{\mathcal{R}}$. There are several elegant ways to prove the Church-Rosser property: among others we could use the historical method of Tait and Martin-Löf (see Definition 3.2.3 of [3]), or the van Oostrom and van Raamsdonk technique [47], or the Takahashi parallel reduction technique [44]. Parallel reduction semantics extends Definition 4 and it is inductively defined as follows.

▶ **Definition 18** (Parallel reduction semantics).

$$\begin{array}{rcll}
x & \Longrightarrow & x & \textit{and} \qquad u_\Delta \Longrightarrow u_\Delta \\
\Delta^{\sigma} & \Longrightarrow & (\Delta')^{\sigma} & \textit{if } \Delta \Longrightarrow \Delta' \\
\Delta_1\, \Delta_2 & \Longrightarrow & \Delta'_1\, \Delta'_2 & \textit{if } \Delta_1 \Longrightarrow \Delta'_1 \textit{ and } \Delta_2 \Longrightarrow \Delta'_2 \\
\lambda x{:}\sigma.\Delta & \Longrightarrow & \lambda x{:}\sigma.\Delta' & \textit{if } \Delta \Longrightarrow \Delta' \\
(\lambda x{:}\sigma.\Delta_1)\, \Delta_2 & \Longrightarrow & \Delta'_1[\Delta'_2/x] & \textit{if } \Delta_1 \Longrightarrow \Delta'_1 \textit{ and } \Delta_2 \Longrightarrow \Delta'_2 \\
\langle \Delta_1 \,, \Delta_2\rangle & \Longrightarrow & \langle \Delta'_1 \,, \Delta'_2\rangle & \textit{if } \Delta_1 \Longrightarrow \Delta'_1 \textit{ and } \Delta_2 \Longrightarrow \Delta'_2 \\
pr_i\, \Delta & \Longrightarrow & pr_i\, \Delta' & \textit{if } \Delta \Longrightarrow \Delta' \quad \textit{and } i \in \{1,2\} \\
pr_i\, \langle \Delta_1 \,, \Delta_2\rangle & \Longrightarrow & \Delta'_i & \textit{if } \Delta_i \Longrightarrow \Delta'_i \textit{ and } i \in \{1,2\}
\end{array}$$

Intuitively, $\Delta \Longrightarrow \Delta'$ means that Δ' is obtained from Δ by simultaneous contraction of some βpr_i-redexes possibly overlapping each other. Church-Rosser can be achieved by proving a stronger statement, namely $\Delta \Longrightarrow \Delta'$ implies $\Delta' \Longrightarrow \Delta^*$, where Δ^* is a Δ-term determined by Δ and independent from Δ'. The above statement is satisfied by the term Δ^* which is, in turn, obtained from Δ by contracting all the redexes existing in Δ simultaneously.

▶ **Definition 19** (The map _*).

$$
\begin{array}{rcl@{\qquad}rcl}
x^* & \stackrel{def}{=} & x & u_\Delta^* & \stackrel{def}{=} & u_\Delta \\
(\Delta^\sigma)^* & \stackrel{def}{=} & (\Delta^*)^\sigma & \langle \Delta_1 \,, \Delta_2 \rangle^* & \stackrel{def}{=} & \langle \Delta_1^* \,, \Delta_2^* \rangle \\
(\lambda x{:}\sigma.\Delta)^* & \stackrel{def}{=} & \lambda x{:}\sigma.\Delta^* & (\lambda x{:}\sigma.\Delta_1)\, \Delta_2^* & \stackrel{def}{=} & \Delta_1^*[\Delta_2^*/x] \\
(\Delta_1\, \Delta_2)^* & \stackrel{def}{=} & \Delta_1^*\, \Delta_2^* & & & \textit{if } \Delta_1\, \Delta_2 \textit{ is not a } \beta\textit{-redex} \\
(pr_i\, \langle \Delta_1 \,, \Delta_2 \rangle)^* & \stackrel{def}{=} & \Delta_i^* & & & i \in \{1, 2\} \\
(pr_i\, \Delta)^* & \stackrel{def}{=} & pr_i\, \Delta^* & & & \textit{if } \Delta \textit{ is not a strong pair}
\end{array}
$$

The next technical lemma will be useful in showing that Church-Rosser for $\longrightarrow\!\!\!\!\!\rightarrow$ can be inherited from Church-Rosser for $\Longrightarrow$.

▶ **Lemma 20.**

1. *If* $\Delta_1 \longrightarrow \Delta_1'$, *then* $\Delta_1 \Longrightarrow \Delta_1'$;
2. *if* $\Delta_1 \Longrightarrow \Delta_1'$, *then* $\Delta_1 \longrightarrow\!\!\!\!\!\rightarrow \Delta_1'$;
3. *if* $\Delta_1 \Longrightarrow \Delta_1'$ *and* $\Delta_2 \Longrightarrow \Delta_2'$, *then* $\Delta_1[\Delta_2/x] \Longrightarrow \Delta_1'[\Delta_2'/x]$;
4. $\Delta_1 \Longrightarrow \Delta_1^*$.

Now we have to prove the Church-Rosser property for the parallel reduction.

▶ **Lemma 21** (Confluence property for $\Longrightarrow$). *If* $\Delta \Longrightarrow \Delta'$, *then* $\Delta' \Longrightarrow \Delta^*$.

The Church-Rosser property follows.

▶ **Theorem 22** (Confluence). *If* $\Delta_1 \longrightarrow\!\!\!\!\!\rightarrow \Delta_2$ *and* $\Delta_1 \longrightarrow\!\!\!\!\!\rightarrow \Delta_3$, *then there exists* Δ_4 *such that* $\Delta_2 \longrightarrow\!\!\!\!\!\rightarrow \Delta_4$ *and* $\Delta_3 \longrightarrow\!\!\!\!\!\rightarrow \Delta_4$.

At this point, a small remark about η-reduction in the Δ-calculus could be useful. It is well-known by Nederpelt [35] that Church-Rosser for $\beta\eta$-reduction in the λ-calculus à la Church is just false; Geuvers in [23] proved the Church-Rosser for $\beta\eta$-reduction for well-typed terms in Pure Type Systems; the same result can be easily proved for the Δ-calculus.

The next lemma says that all type derivations for Δ have an unique type.

▶ **Lemma 23** (Unicity of typing). *If* $B \vdash_{\mathcal{R}}^{\mathcal{T}} \Delta : \sigma$, *then* σ *is unique.*

The next lemma proves inversion properties on typable Δ-terms.

▶ **Lemma 24** (Generation).

1. *If* $B \vdash_{\mathcal{R}}^{\mathcal{T}} x : \sigma$, *then* $x{:}\sigma \in B$;
2. *if* $B \vdash_{\mathcal{R}}^{\mathcal{T}} \lambda x{:}\sigma.\Delta : \rho$, *then* $\rho \equiv \sigma \to \tau$ *for some* τ *and* $B, x{:}\sigma \vdash_{\mathcal{R}}^{\mathcal{T}} \Delta : \tau$;
3. *if* $B \vdash_{\mathcal{R}}^{\mathcal{T}} \Delta_1\, \Delta_2 : \tau$, *then there is* σ *such that* $B \vdash_{\mathcal{R}}^{\mathcal{T}} \Delta_1 : \sigma \to \tau$ *and* $B \vdash_{\mathcal{R}}^{\mathcal{T}} \Delta_2 : \sigma$;
4. *if* $B \vdash_{\mathcal{R}}^{\mathcal{T}} \langle \Delta_1 \,, \Delta_2 \rangle : \rho$, *then there is* σ, τ *such that* $\rho \equiv \sigma \cap \tau$ *and* $B \vdash_{\mathcal{R}}^{\mathcal{T}} \Delta_1 : \sigma$ *and* $B \vdash_{\mathcal{R}}^{\mathcal{T}} \Delta_2 : \tau$ *and* $\wr \Delta_1 \wr \mathcal{R} \wr \Delta_2 \wr$;
5. *if* $B \vdash_{\mathcal{R}}^{\mathcal{T}} pr_1\, \Delta : \sigma$, *then there is* τ *such that* $B \vdash_{\mathcal{R}}^{\mathcal{T}} \Delta : \sigma \cap \tau$;
6. *if* $B \vdash_{\mathcal{R}}^{\mathcal{T}} pr_2\, \Delta : \tau$, *then there is* σ *such that* $B \vdash_{\mathcal{R}}^{\mathcal{T}} \Delta : \sigma \cap \tau$;
7. *if* $B \vdash_{\mathcal{R}}^{\mathcal{T}} u_\Delta : \sigma$, *then* $\sigma \equiv \mathtt{U}$;
8. *if* $B \vdash_{\mathcal{R}}^{\mathcal{T}} \Delta^\tau : \rho$, *then* $\rho \equiv \tau$ *and there is* σ *such that* $\sigma \leqslant_{\mathcal{T}} \tau$ *and* $B \vdash_{\mathcal{R}}^{\mathcal{T}} \Delta : \sigma$.

The next lemma says that all subterms of a typable Δ-term are typable too.

▶ **Lemma 25** (Subterms typability). *If $B \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta : \sigma$, and Δ' is a subterm of Δ, then there exists B' and τ such that $B' \supseteq B$ and $B' \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta' : \tau$.*

As expected, the weakening and strengthening properties on contexts are verified.

▶ **Lemma 26** (Free-variable properties).
1. *If $B \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta : \sigma$, and $B' \supseteq B$, then $B' \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta : \sigma$;*
2. *if $B \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta : \sigma$, then $\mathrm{FV}(\Delta) \subseteq Dom(B)$;*
3. *if $B \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta : \sigma$, $B' \subseteq B$ and $\mathrm{FV}(\Delta) \subseteq Dom(B')$, then $B' \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta : \sigma$.*

The next lemma also says that essence is closed under substitution.

▶ **Lemma 27** (Substitution).
1. $\wr \Delta_1[\Delta_2/x] \wr \equiv \wr \Delta_1 \wr [\wr \Delta_2 \wr /x]$;
2. *If $B, x{:}\sigma \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta_1 : \tau$ and $B \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta_2 : \sigma$, then $B \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta_1[\Delta_2/x] : \tau$.*

In order to prove subject reduction, we need to prove that reducing Δ-terms preserve the side-condition $\wr \Delta_1 \wr \mathcal{R} \wr \Delta_2 \wr$ when typing the strong pair $\langle \Delta_1 , \Delta_2 \rangle$. We prove this in the following lemma.

▶ **Lemma 28** (Essence reduction).
1. *If $B \vdash^{\mathcal{T}}_{\equiv} \Delta_1 : \sigma$ and $\Delta_1 \longrightarrow \Delta_2$, then $\wr \Delta_1 \wr =_{\beta} \wr \Delta_2 \wr$;*
2. *for $\mathcal{R} \in \{=_{\beta}, =_{\beta\eta}\}$, if $B \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta_1 : \sigma$ and $\Delta_1 \longrightarrow \Delta_2$, then $\wr \Delta_1 \wr \mathcal{R} \wr \Delta_2 \wr$;*
3. *if $B \vdash^{\mathcal{T}}_{=_{\beta\eta}} \Delta_1 : \sigma$ and $\Delta_1 \longrightarrow_{\eta} \Delta_2$, then $\wr \Delta_1 \wr =_{\eta} \wr \Delta_2 \wr$.*

The next theorem states that all the $\Delta^{\mathcal{T}}_{\equiv}$ typed systems preserve synchronous βpr_i-reduction, and all the $\Delta^{\mathcal{T}}_{=_{\beta}}$ and $\Delta^{\mathcal{T}}_{=_{\beta\eta}}$ typed systems preserve βpr_i-reduction.

▶ **Theorem 29** (Subject reduction for βpr_i).
1. *If $B \vdash^{\mathcal{T}}_{\equiv} \Delta_1 : \sigma$ and $\Delta_1 \longrightarrow^{\parallel} \Delta_2$, then $B \vdash^{\mathcal{T}}_{\equiv} \Delta_2 : \sigma$;*
2. *for $\mathcal{R} \in \{=_{\beta}, =_{\beta\eta}\}$, if $B \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta_1 : \sigma$ and $\Delta_1 \longrightarrow \Delta_2$, then $B \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta_2 : \sigma$.*

The next theorem states that some of the typed systems on the back of the Δ-chair preserve η-reduction.

▶ **Theorem 30** (Subject reduction for η for $\mathcal{T}_{\mathrm{CDV}}, \mathcal{T}_{\mathrm{BCD}}$). *Let $\mathcal{T} \in \{\mathcal{T}_{\mathrm{CDV}}, \mathcal{T}_{\mathrm{BCD}}\}$. If $B \vdash^{\mathcal{T}}_{=_{\beta\eta}} \Delta_1 : \sigma$ and $\Delta_1 \longrightarrow_{\eta} \Delta_2$, then $B \vdash^{\mathcal{T}}_{=_{\beta\eta}} \Delta_2 : \sigma$.*

4.2 Strong normalization

The idea of the strong normalization proof is to embed typable terms of the Δ-calculus into Church-style terms of a target system, which is the simply-typed λ-calculus with pairs, in a structure-preserving way (and forgetting all the essence side-conditions). The translation is sufficiently faithful so as to preserve the number of reductions, and so strong normalization for the Δ-calculus follows from strong normalization for simply-typed λ-calculus with pairs. A similar technique has been used in [24] to prove the strong normalization property of LF and in [9] to prove the strong normalization property of a subset of $\lambda^{\mathrm{CD}}_{\cap}$.

The target system has one atomic type called $\circ$, a special constant term $u_{\circ}$ of type $\circ$ and an infinite number of constants c_{σ} of type σ for any type of the target system. We denote by $B \vdash_{\times} M : \sigma$ a typing judgment in the target system.

▶ **Definition 31** (Forgetful mapping).

- *On intersection types.*

$$|a_i| \stackrel{def}{=} \circ \quad \forall a_i \in \mathbb{A} \qquad and \qquad |\sigma \cap \tau| \stackrel{def}{=} |\sigma| \times |\tau| \qquad and \qquad |\sigma \to \tau| \stackrel{def}{=} |\sigma| \to |\tau|$$

- *On Δ-terms.*

$$\begin{array}{rcl} |x|_B & \stackrel{def}{=} & x \\ |\lambda x{:}\sigma.\Delta|_B & \stackrel{def}{=} & \lambda x.|\Delta|_{B,x:\sigma} \\ |\langle \Delta_1 \,, \Delta_2 \rangle|_B & \stackrel{def}{=} & (|\Delta_1|_B \,, |\Delta_2|_B) \\ |\Delta^\tau|_B & \stackrel{def}{=} & c_{|\sigma|\to|\tau|}\,|\Delta|_B \quad \text{if } B \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta : \sigma \end{array} \qquad \begin{array}{rcl} |u_\Delta|_B & \stackrel{def}{=} & u_\circ \\ |\Delta_1\,\Delta_2|_B & \stackrel{def}{=} & |\Delta_1|_B\,|\Delta_2|_B \\ |pr_i\,\Delta|_B & \stackrel{def}{=} & pr_i\,|\Delta|_B \end{array}$$

- *The map can be easily extended to basis B.*

The following technical lemma states some properties of the forgetful function.

▶ **Lemma 32.**

1. *If $B \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta : \sigma$, then $|\Delta|_B$ is defined, and, for all $B' \supseteq B$, $|\Delta|_B \equiv |\Delta|_{B'}$;*
2. *$|\Delta_1[\Delta_2/x]|_B \equiv |\Delta_1|_B[|\Delta_2|_B/x]$;*
3. *If $\Delta_1 \longrightarrow \Delta_2$, then $|\Delta_1|_B \longrightarrow |\Delta_2|_B$;*
4. *If $B \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta : \sigma$ then $|B| \vdash_\times |\Delta|_B : |\sigma|$.*

Strong normalization follows easily from the above lemmas.

▶ **Theorem 33** (Strong normalization). *If $B \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta : \sigma$, then Δ is strongly normalizing.*

5 Typed systems à la Church vs. type assignment systems à la Curry

5.1 Relation between type assignment systems $\lambda^{\mathcal{T}}_\cap$ and typed systems $\Delta^{\mathcal{T}}_{\mathcal{R}}$

It is interesting to state some relations between type assignment systems à la Church and typed systems à la Curry. An interesting property is the one of isomorphism, namely the fact that whenever we assign a type σ to a pure λ-term M, the same type can be assigned to a Δ-term such that the essence of Δ is M. Conversely, for every assignment of σ to a Δ-term, a valid type assignment judgment of the same type for the essence of Δ can be derived. Soundness, completeness and isomorphism between intersection typed systems for the Δ-calculus and the corresponding intersection type assignment systems for the λ-calculus are defined as follows.

▶ **Definition 34** (Soundness, completeness and isomorphism). *Let $\Delta^{\mathcal{T}}_{\mathcal{R}}$ and $\lambda^{\mathcal{T}}_\cap$.*

1. *(Soundness, $\Delta^{\mathcal{T}}_{\mathcal{R}} \lhd \lambda^{\mathcal{T}}_\cap$). $B \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta : \sigma$ implies $B \vdash^{\mathcal{T}}_\cap \wr \Delta \wr : \sigma$;*
2. *(Completeness, $\Delta^{\mathcal{T}}_{\mathcal{R}} \rhd \lambda^{\mathcal{T}}_\cap$). $B \vdash^{\mathcal{T}}_\cap M : \sigma$ implies there exists Δ such that $M \equiv \wr \Delta \wr$ and $B \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta : \sigma$;*
3. *(Isomorphism, $\Delta^{\mathcal{T}}_{\mathcal{R}} \sim \lambda^{\mathcal{T}}_\cap$). $\Delta^{\mathcal{T}}_{\mathcal{R}} \rhd \lambda^{\mathcal{T}}_\cap$ and $\Delta^{\mathcal{T}}_{\mathcal{R}} \lhd \lambda^{\mathcal{T}}_\cap$.*

▶ **Theorem 35** (Soundness, completeness and isomorphism). *The following properties (left of Figure 4) between Δ-calculi and type assignment systems $\lambda^{\mathcal{T}}_\cap$ can be verified.*

The next theorem characterizes the class of strongly normalizing Δ-terms.

$\Delta^{\mathcal{T}}_{\mathcal{R}}$	$\Delta^{\mathcal{T}}_{\mathcal{R}} \lhd \lambda^{\mathcal{T}}_{\cap}$	$\Delta^{\mathcal{T}}_{\mathcal{R}} \rhd \lambda^{\mathcal{T}}_{\cap}$
$\Delta^{\mathrm{CD}}_{\equiv}$	$\surd$	$\surd$
$\Delta^{\mathrm{CDV}}_{\equiv}$	$\surd$	$\surd$
$\Delta^{\mathrm{CDS}}_{\equiv}$	$\surd$	$\surd$
$\Delta^{\mathrm{BCD}}_{\equiv}$	$\surd$	$\surd$
$\Delta^{\mathrm{CD}}_{=_\beta}$	$\times$	$\surd$
$\Delta^{\mathrm{CDV}}_{=_\beta}$	$\times$	$\surd$
$\Delta^{\mathrm{CDS}}_{=_\beta}$	$\surd$	$\surd$
$\Delta^{\mathrm{BCD}}_{=_\beta}$	$\surd$	$\surd$
$\Delta^{\mathrm{CDV}}_{=_{\beta\eta}}$	$\times$	$\surd$
$\Delta^{\mathrm{BCD}}_{=_{\beta\eta}}$	$\times$	$\surd$

$\Delta^{\mathcal{T}}_{\mathcal{R}}$	TC/TR
$\Delta^{\mathrm{CD}}_{\equiv}$	$\surd$
$\Delta^{\mathrm{CDV}}_{\equiv}$	$\surd$
$\Delta^{\mathrm{CDS}}_{\equiv}$	$\surd$
$\Delta^{\mathrm{BCD}}_{\equiv}$	$\surd$
$\Delta^{\mathrm{CD}}_{=_\beta}$	$\surd$
$\Delta^{\mathrm{CDV}}_{=_\beta}$	$\surd$
$\Delta^{\mathrm{CDS}}_{=_\beta}$	$\times$
$\Delta^{\mathrm{BCD}}_{=_\beta}$	$\times$
$\Delta^{\mathrm{CDV}}_{=_{\beta\eta}}$	$\surd$
$\Delta^{\mathrm{BCD}}_{=_{\beta\eta}}$	$\times$

Source	Target
$\Delta^{\mathrm{CD}}_{\equiv}$	$\Delta^{\mathrm{CD}}_{=_\beta}$
$\Delta^{\mathrm{CDV}}_{\equiv}$	$\Delta^{\mathrm{CDV}}_{=_{\beta\eta}}$
$\Delta^{\mathrm{CDS}}_{\equiv}$	$\Delta^{\mathrm{CDS}}_{=_\beta}$
$\Delta^{\mathrm{BCD}}_{\equiv}$	$\Delta^{\mathrm{BCD}}_{=_{\beta\eta}}$
$\Delta^{\mathrm{CD}}_{=_\beta}$	$\Delta^{\mathrm{CD}}_{=_\beta}$
$\Delta^{\mathrm{CDV}}_{=_\beta}$	$\Delta^{\mathrm{CDV}}_{=_{\beta\eta}}$
$\Delta^{\mathrm{CDS}}_{=_\beta}$	$\Delta^{\mathrm{CDS}}_{=_\beta}$
$\Delta^{\mathrm{BCD}}_{=_\beta}$	$\Delta^{\mathrm{BCD}}_{=_{\beta\eta}}$
$\Delta^{\mathrm{CDV}}_{=_{\beta\eta}}$	$\Delta^{\mathrm{CDV}}_{=_{\beta\eta}}$
$\Delta^{\mathrm{BCD}}_{=_{\beta\eta}}$	$\Delta^{\mathrm{BCD}}_{=_{\beta\eta}}$

Figure 4 On the left: Soundness, completeness, isomorphism. On the center: type checking/reconstruction. On the right: source and target languages of the translation.

▶ **Theorem 36** (Characterization). *Every strongly normalizing λ-term can be type-annotated so as to be the essence of a typable Δ-term.*

We can finally state decidability of type checking (TC) and type reconstruction (TR).

▶ **Theorem 37** (Decidability of type checking and type reconstruction). *Figure 4 (in the center) list decidability of type checking and type reconstruction.*

5.2 Subtyping and explicit coercions

The typing rule $(\leqslant_{\mathcal{T}})$ in the general typed system introduces type coercions: once a type coercion is introduced, it cannot be eliminated, so *de facto* freezing a Δ-term inside an explicit coercion. Tannen et al. [8] showed a translation of a judgment derivation from a "Source" system with subtyping (Cardelli's Fun [11]) into an "equivalent" judgment derivation in a "Target" system without subtyping (Girard system F with records and recursion). In the same spirit, we present a translation that removes all explicit coercions. Intuitively, the translation proceeds as follows: every derivation ending with rule $(\leqslant_{\mathcal{T}})$ is translated into the following (coercion-free) derivation, i.e.

$$\frac{B \vdash^{\mathcal{T}}_{\mathcal{R}'} \|\sigma \leqslant_{\mathcal{T}} \tau\| : \sigma \to \tau \qquad B \vdash^{\mathcal{T}}_{\mathcal{R}'} \|\Delta\|_B : \sigma}{B \vdash^{\mathcal{T}}_{\mathcal{R}'} \|\sigma \leqslant_{\mathcal{T}} \tau\| \; \|\Delta\|_B : \tau} \; (\to E)$$

where $\mathcal{R}'$ is a suitable relation such that $\mathcal{R} \sqsubseteq \mathcal{R}'$, $\|-\|$ is a suitable translation function defined later in Definition 39. Note that changing of the type theory is necessary to guarantee well-typedness in the translation of strong pairs. Summarizing, we provide a type preserving translation of a Δ-term into a coercion-free Δ-term such that $\wr\,\Delta\,\wr =_{\beta\eta} \wr\,\Delta'\,\wr$. The following example illustrates some trivial compilations of axioms and rule schemes of Figure 1.

▶ **Example 38** (Translation of axioms and rule schemes of Figure 1).

(refl) the judgment $x{:}\sigma \vdash^{\mathcal{T}}_{\mathcal{R}} \langle x \,, x^{\sigma}\rangle : \sigma \cap \sigma$ is translated to a coercion-free judgment $x{:}\sigma \vdash^{\mathcal{T}}_{=_\beta} \langle x \,, (\lambda y{:}\sigma.y)\, x\rangle : \sigma \cap \sigma$;

(incl) the judgment $x{:}\sigma \cap \tau \vdash^{\mathcal{T}}_{\mathcal{R}} \langle x \,, x^{\tau}\rangle : (\sigma \cap \tau) \cap \tau$ is translated to a coercion-free judgment $x{:}\sigma \cap \tau \vdash^{\mathcal{T}}_{=_\beta} \langle x \,, (\lambda y{:}\sigma \cap \tau.pr_2\, y)\, x\rangle : (\sigma \cap \tau) \cap \tau$;

(glb) the judgment $x{:}\sigma \vdash^{\mathcal{T}}_{\mathcal{R}} \langle x \,, x^{\sigma\cap\sigma}\rangle : \sigma \cap (\sigma \cap \sigma)$ is translated to a coercion-free judgment $x{:}\sigma \vdash^{\mathcal{T}}_{=_\beta} \langle x \,, (\lambda y{:}\sigma.\langle y \,, y\rangle)\, x\rangle : \sigma \cap (\sigma \cap \sigma)$;

($\mathtt{U}_{top}$) the judgment $x{:}\sigma \vdash^{\mathcal{T}}_{\mathcal{R}} \langle x \,, x^{\mathtt{U}}\rangle : \sigma \cap \mathtt{U}$ is translated to a coercion-free judgment $x{:}\sigma \vdash^{\mathcal{T}}_{=_\beta} \langle x \,, (\lambda y{:}\sigma.u_y)\, x\rangle : \sigma \cap \mathtt{U}$;

($\mathtt{U}_{\to}$) the judgment $x{:}\mathtt{U} \vdash^{\mathcal{T}}_{\mathcal{R}} \langle x \,, x^{\sigma\to\mathtt{U}}\rangle : \mathtt{U} \cap (\sigma \to \mathtt{U})$ is translated to a coercion-free judgment $x{:}\mathtt{U} \vdash^{\mathcal{T}}_{=_{\beta\eta}} \langle x \,, (\lambda f{:}\mathtt{U}.\lambda y{:}\sigma.u_{(f\,y)})\, x\rangle : \mathtt{U} \cap (\sigma \to \mathtt{U})$;

($\to\cap$) the judgment $x{:}(\sigma \to \tau) \cap (\sigma \to \rho) \vdash^{\mathcal{T}}_{\mathcal{R}} x^{\sigma\to\tau\cap\rho} : \sigma \to \tau \cap \rho$ is translated to a coercion-free judgment
$x{:}(\sigma \to \tau)\cap(\sigma \to \rho) \vdash^{\mathcal{T}}_{=_{\beta\eta}} (\lambda f{:}(\sigma \to \tau)\cap(\sigma \to \rho).\lambda y{:}\sigma.\langle (pr_1\, f)\, y \,, (pr_2\, f)\, y\rangle)\, x : \sigma \to \tau\cap\rho$;

($\to$) the judgment $x{:}\sigma \to \tau \cap \rho \vdash^{\mathcal{T}}_{\mathcal{R}} \langle x \,, x^{\sigma\cap\rho\to\tau}\rangle : (\sigma \to \tau \cap \rho) \cap (\sigma \cap \rho \to \tau)$ is translated to a coercion-free judgment
$x{:}\sigma \to \tau\cap\rho \vdash^{\mathcal{T}}_{=_{\beta\eta}} \langle x, (\lambda f{:}\sigma \to \tau \cap \rho.\lambda y{:}\sigma \cap \rho.pr_1\,(f\,(pr_1\, y)))\, x\rangle : (\sigma \to \tau\cap\rho)\cap(\sigma\cap\rho \to \tau)$;

(trans) the judgment $x{:}\sigma \vdash^{\mathcal{T}}_{\mathcal{R}} \langle x \,, (x^{\mathtt{U}})^{\sigma\to\mathtt{U}}\rangle : \sigma \cap (\sigma \to \mathtt{U})$ is translated to a coercion-free judgment
$x{:}\sigma \vdash^{\mathcal{T}}_{=_{\beta\eta}} \langle x \,, (\lambda f{:}\mathtt{U}.\lambda y{:}\sigma.u_{(f\,y)})\, ((\lambda y{:}\sigma.u_y)\, x)\rangle : \sigma \cap (\sigma \to \mathtt{U})$.

The next definition introduces two maps translating subtype judgments into explicit coercions functions and Δ-terms into coercion-free Δ-terms.

▶ **Definition 39** (Translations $\|-\|$ and $\|-\|_B$).

1. *The minimal type theory $\leqslant_{min}$ and the extra axioms and schemes are translated as follows.*

$$\begin{array}{lrcl}
(\text{refl}) & \|\sigma \leqslant_{\mathcal{T}} \sigma\| & \stackrel{def}{=} & \vdash^{\mathcal{T}}_{=_\beta} \lambda x{:}\sigma.x : \sigma \to \sigma \\
(\text{incl}_1) & \|\sigma \cap \tau \leqslant_{\mathcal{T}} \sigma\| & \stackrel{def}{=} & \vdash^{\mathcal{T}}_{=_\beta} \lambda x{:}\sigma \cap \tau.pr_1\, x : \sigma \cap \tau \to \sigma \\
(\text{incl}_2) & \|\sigma \cap \tau \leqslant_{\mathcal{T}} \tau\| & \stackrel{def}{=} & \vdash^{\mathcal{T}}_{=_\beta} \lambda x{:}\sigma \cap \tau.pr_2\, x : \sigma \cap \tau \to \tau \\
(\text{glb}) & \left\|\dfrac{\rho \leqslant_{\mathcal{T}} \sigma \quad \rho \leqslant_{\mathcal{T}} \tau}{\rho \leqslant_{\mathcal{T}} \sigma \cap \tau}\right\| & \stackrel{def}{=} & \vdash^{\mathcal{T}}_{=_\beta} \lambda x{:}\rho.\langle \|\rho \leqslant_{\mathcal{T}} \sigma\|\, x \,, \|\rho \leqslant_{\mathcal{T}} \tau\|\, x\rangle : \rho \to \sigma \cap \tau \\
(\text{trans}) & \left\|\dfrac{\sigma \leqslant_{\mathcal{T}} \tau \quad \tau \leqslant_{\mathcal{T}} \rho}{\sigma \leqslant_{\mathcal{T}} \rho}\right\| & \stackrel{def}{=} & \vdash^{\mathcal{T}}_{=_\beta} \lambda x{:}\sigma.\, \|\tau \leqslant_{\mathcal{T}} \rho\|\, (\|\sigma \leqslant_{\mathcal{T}} \tau\|\, x) : \sigma \to \rho \\
(\mathtt{U}_{top}) & \|\sigma \leqslant_{\mathcal{T}} \mathtt{U}\| & \stackrel{def}{=} & \vdash^{\mathcal{T}}_{=_\beta} \lambda x{:}\sigma.u_x : \sigma \to \mathtt{U}
\end{array}$$

$$(\mathtt{U}_{\to}) \quad \|\mathtt{U} \leqslant_{\mathcal{T}} \sigma \to \mathtt{U}\| \stackrel{def}{=} \vdash^{\mathcal{T}}_{=_{\beta\eta}} \lambda f{:}\mathtt{U}.\lambda x{:}\sigma.u_{(f\,x)} : \mathtt{U} \to (\sigma \to \mathtt{U})$$

Let $\xi_1 \stackrel{def}{=} (\sigma \to \tau) \cap (\sigma \to \rho)$ and $\xi_2 \stackrel{def}{=} \sigma \to \tau \cap \rho$

$$(\to\cap) \quad \|\xi_1 \leqslant_{\mathcal{T}} \xi_2\| \stackrel{def}{=} \vdash^{\mathcal{T}}_{=_{\beta\eta}} \lambda f{:}\xi_1.\lambda x{:}\sigma.\langle (pr_1\, f)\, x \,, (pr_2\, f)\, x\rangle : \xi_1 \to \xi_2$$

Let $\xi_1 \stackrel{def}{=} \sigma_1 \to \tau_1$ and $\xi_2 \stackrel{def}{=} \sigma_2 \to \tau_2$

$$(\to) \quad \left\|\dfrac{\sigma_2 \leqslant_{\mathcal{T}} \sigma_1 \quad \tau_1 \leqslant_{\mathcal{T}} \tau_2}{\sigma_1 \to \tau_1 \leqslant_{\mathcal{T}} \sigma_2 \to \tau_2}\right\| \stackrel{def}{=} \vdash^{\mathcal{T}}_{=_{\beta\eta}} \lambda f{:}\xi_1.\lambda x{:}\sigma_2.\, \|\tau_1 \leqslant_{\mathcal{T}} \tau_2\|\, (f\,(\|\sigma_2 \leqslant_{\mathcal{T}} \sigma_1\|\, x)) : \xi_1 \to \xi_2$$

2. *The translation $\|-\|_B$ is defined on Δ as follows.*

$$\begin{array}{rclrcll}
\|u_\Delta\|_B & \stackrel{def}{=} & u_{\|\Delta\|_B} & \|x\|_B & \stackrel{def}{=} & x & \\
\|\lambda x{:}\sigma.\Delta\|_B & \stackrel{def}{=} & \lambda x{:}\sigma.\, \|\Delta\|_{B,x:\sigma} & \|\Delta_1\, \Delta_2\|_B & \stackrel{def}{=} & \|\Delta_1\|_B\, \|\Delta_2\|_B & \\
\|\langle \Delta_1 \,, \Delta_2\rangle\|_B & \stackrel{def}{=} & \langle \|\Delta_1\|_B \,, \|\Delta_2\|_B\rangle & \|pr_i\, \Delta\|_B & \stackrel{def}{=} & pr_i\, \|\Delta\|_B & i \in \{1,2\} \\
\|\Delta^\tau\|_B & \stackrel{def}{=} & \|\sigma \leqslant_{\mathcal{T}} \tau\|\, \|\Delta\|_B & \multicolumn{3}{l}{\textit{if } B \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta : \sigma.} &
\end{array}$$

By looking at the above translation functions we can see that if $B \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta : \sigma$, then $\|\Delta\|_B$ is defined and it is coercion-free.

The following key lemma states that a coercion function is always typable in $\Delta^{\mathcal{T}}_{=_{\beta\eta}}$, that it is essentially the identity and that, without using the rule schemes $(\to\cap)$, $(\mathtt{U}_\to)$, and $(\to)$ the translation can even be derivable in $\Delta^{\mathcal{T}}_{=_\beta}$.

▶ **Lemma 40** (Essence of a coercion is an identity).

1. *If $\sigma \leqslant_{\mathcal{T}} \tau$, then $\vdash^{\mathcal{T}}_{=_{\beta\eta}} \|\sigma \leqslant_{\mathcal{T}} \tau\| : \sigma \to \tau$ and $\wr \|\sigma \leqslant_{\mathcal{T}} \tau\| \wr =_{\beta\eta} \lambda x.x$;*
2. *If $\sigma \leqslant_{\mathcal{T}} \tau$ without using the rule schemes $(\to\cap)$, $(\mathtt{U}_\to)$, and $(\to)$, then $\vdash^{\mathcal{T}}_{=_\beta} \|\sigma \leqslant_{\mathcal{T}} \tau\| : \sigma \to \tau$ and $\wr \|\sigma \leqslant_{\mathcal{T}} \tau\| \wr =_\beta \lambda x.x$.*

Proof. The proofs proceed in both parts by induction on the derivation of $\sigma \leqslant_{\mathcal{T}} \tau$. For instance, in case of (glb), we can verify that $\vdash^{\mathcal{T}}_{=_\beta} \lambda x{:}\rho.\langle \|\rho \leqslant_{\mathcal{T}} \sigma\|\, x \,, \|\rho \leqslant_{\mathcal{T}} \tau\|\, x\rangle : \rho \to \sigma \cap \tau$ using the induction hypotheses that $\|\rho \leqslant_{\mathcal{T}} \sigma\|$ (resp. $\|\rho \leqslant_{\mathcal{T}} \tau\|$) has type $\rho \to \sigma$ (resp. $\rho \to \tau$) and has an essence convertible to $\lambda x.x$. ◀

We can now prove the coherence of the translation as follows.

▶ **Theorem 41** (Coherence). *If $B \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta : \sigma$, then $B \vdash^{\mathcal{T}}_{\mathcal{R}'} \|\Delta\|_B : \sigma$ and $\wr \|\Delta\|_B \wr \ \mathcal{R}' \wr \Delta \wr$, where $\Delta^{\mathcal{T}}_{\mathcal{R}}$ and $\Delta^{\mathcal{T}}_{\mathcal{R}'}$ are respectively the source and target intersection typed systems given in Figure 4 (right part).*

Proof. By induction on the derivation. We illustrate the most important case, namely when the last type rule is $(\leqslant_{\mathcal{T}})$. In this case $\|\Delta^\tau\|_B$ is translated to $\|\sigma \leqslant_{\mathcal{T}} \tau\|\ \|\Delta\|_B$. By induction hypothesis we have that $B \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta : \sigma$, and by Lemma 40 we have that $B \vdash^{\mathcal{T}}_{\mathcal{R}'} \|\sigma \leqslant_{\mathcal{T}} \tau\| : \sigma \to \tau$; therefore $B \vdash^{\mathcal{T}}_{\mathcal{R}'} \|\Delta^\tau\|_B : \tau$. Moreover, we know that $\wr \|\sigma \leqslant_{\mathcal{T}} \tau\| \wr \ \mathcal{R}'\ \lambda x.x$, and this gives $\wr \|\Delta^\tau\|_B \wr \ \mathcal{R}' \wr \|\Delta\|_B \wr$. Again by induction hypothesis we have that $\wr \|\Delta\|_B \wr \ \mathcal{R}' \wr \Delta \wr$, and this gives the thesis $\wr \|\Delta^\tau\|_B \wr \ \mathcal{R}' \wr \Delta^\tau \wr$. ◀

References

1. Fabio Alessi, Franco Barbanera, and Mariangiola Dezani-Ciancaglini. Intersection types and lambda models. *Theor. Comput. Sci.*, 355(2):108–126, 2006.
2. Franco Barbanera and Simone Martini. Proof-functional connectives and realizability. *Archive for Mathematical Logic*, 33:189–211, 1994.
3. Henk Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, revised edition, 1984.
4. Henk P. Barendregt, Mario Coppo, and Mariangiola Dezani-Ciancaglini. A Filter lambda model and the completeness of type assignment. *Journal of Symbolic Logic*, 48(4):931–940, 1983.
5. Henk P. Barendregt, Wil Dekkers, and Richard Statman. *Lambda calculus with types.* Cambridge University Press, 2013.
6. Choukri-Bey Ben-Yelles. *Type assignment in the lambda-calculus: syntax and semantics.* PhD thesis, University College of Swansea, 1979.
7. Viviana Bono, Betti Venneri, and Lorenzo Bettini. A typed lambda calculus with intersection types. *Theor. Comput. Sci.*, 398(1-3):95–113, 2008.
8. Val Breazu-Tannen, Thierry Coquand, Carl A. Gunter, and Andre Scedrov. Inheritance as Implicit Coercion. *Inf. Comput.*, 93(1):172–221, 1991.
9. Antonio Bucciarelli, Adolfo Piperno, and Ivano Salvo. Intersection types and λ-definability. *Mathematical Structures in Computer Science*, 13(1):15–53, 2003.
10. Beatrice Capitani, Michele Loreti, and Betti Venneri. Hyperformulae, Parallel Deductions and Intersection Types. *Proc. of BOTH, Electr. Notes Theor. Comput. Sci.*, 50(2):180–198, 2001.

11 Luca Cardelli and Peter Wegner. On Understanding types, data abstraction, and polymorphism. *ACM Computing Surveys*, 17(4):471–523, December 1985.

12 Mario Coppo and Mariangiola Dezani-Ciancaglini. An Extension of the Basic Functionality Theory for the λ-calculus. *Notre Dame Journal of Formal Logic*, 21(4):685–693, 1980.

13 Mario Coppo, Mariangiola Dezani-Ciancaglini, and Patrick Sallé. Functional characterization of some semantic equalities inside λ-calculus. In *International Colloquium on Automata, Languages, and Programming*, pages 133–146. Springer-Verlag, 1979.

14 Mario Coppo, Mariangiola Dezani-Ciancaglini, and Betti Venneri. Functional characters of solvable terms. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 27(2-6):45–58, 1981.

15 Rowan Davies. *Practical Refinement-Type Checking*. PhD thesis, Carnegie Mellon University, 2005. CMU-CS-05-110.

16 Daniel J. Dougherty, Ugo de'Liguoro, Luigi Liquori, and Claude Stolze. A Realizability Interpretation for Intersection and Union Types. In *Proc. of APLAS*, volume 10017 of *Lecture Notes in Computer Science*, pages 187–205. Springer-Verlag, 2016.

17 Daniel J. Dougherty and Luigi Liquori. Logic and Computation in a Lambda Calculus with Intersection and Union Types. In *Proc. of LPAR*, volume 6355 of *Lecture Notes in Computer Science*, pages 173–191. Springer-Verlag, 2010.

18 Andrej Dudenhefner, Moritz Martens, and Jakob Rehof. The Algebraic Intersection Type Unification Problem. *Logical Methods in Computer Science*, 13(3), 2017.

19 Joshua Dunfield. Refined typechecking with Stardust. In *Proc. of PLPV*, pages 21–32, 2007.

20 Joshua Dunfield. Elaborating intersection and union types. *J. Funct. Program.*, 24(2-3):133–165, 2014.

21 Timothy S. Freeman and Frank Pfenning. Refinement Types for ML. In *Proc. of PLDI*, pages 268–277, 1991.

22 Alain Frisch, Giuseppe Castagna, and Véronique Benzaken. Semantic subtyping: Dealing set-theoretically with function, union, intersection, and negation types. *Journal of the ACM*, 55(4):19, 2008.

23 Herman Geuvers. The Church-Rosser Property for beta-eta-reduction in Typed lambda-Calculi. In *Proc. of LICS*, pages 453–460, 1992.

24 Robert Harper, Furio Honsell, and Gordon Plotkin. A Framework for Defining Logics. *J. ACM*, 40(1):143–184, 1993.

25 J. Roger Hindley. The simple semantics for Coppo-Dezani-Sallé types. In *International Symposium on Programming*, pages 212–226, 1982.

26 J. Roger Hindley. Coppo-Dezani types do not correspond to propositional logic. *Theor. Comput. Sci.*, 28:235–236, 1984.

27 Furio Honsell, Luigi Liquori, Claude Stolze, and Ivan Scagnetto. The Delta-Framework. In *Proc of FSTTCS*, pages 37:1–37:21, 2018.

28 Assaf J. Kfoury and Joe B. Wells. Principality and type inference for intersection types using expansion variables. *Theor. Comput. Sci.*, 311(1-3):1–70, 2004.

29 Luigi Liquori and Simona Ronchi Della Rocca. Towards an intersection typed system *à la* Church. *Proc. of ITRS, Electronic Notes in Theoretical Computer Science*, 136:43–56, 2005.

30 Luigi Liquori and Simona Ronchi Della Rocca. Intersection Typed System *à la* Church. *Information and Computation*, 9(205):1371–1386, 2007.

31 Luigi Liquori and Claude Stolze. A Decidable Subtyping Logic for Intersection and Union Types. In *Proc of TTCS*, volume 10608 of *Lecture Notes in Computer Science*, pages 74–90. Springer-Verlag, 2017.

32 Edgar G. K. Lopez-Escobar. Proof functional connectives. In *Methods in Mathematical Logic*, volume 1130 of *Lecture Notes in Mathematics*, pages 208–221. Springer-Verlag, 1985.

33 Grigori Mints. The Completeness of Provable Realizability. *Notre Dame Journal of Formal Logic*, 30(3):420–441, 1989.

34 Alexandre Miquel. The Implicit Calculus of Constructions. In *Proc. of TLCA*, volume 2044 of *Lecture Notes in Computer Science*, pages 344–359. Springer-Verlag, 2001.

35 Rob. P. Nederpelt. *Strong Normalization in a typed lambda calculus with lambda structured types.* PhD thesis, Eindhoven Technological University, 1973.

36 Benjamin C. Pierce. *Programming with intersection types, union types, and bounded polymorphism.* PhD thesis, Technical Report CMU-CS-91-205. Carnegie Mellon University, 1991.

37 Benjamin C. Pierce. Programming with intersection types, union types, and polymorphism. Technical Report CMU-CS-91-106, Carnegie Mellon University, 1991.

38 Elaine Pimentel, Simona Ronchi Della Rocca, and Luca Roversi. Intersection Types from a Proof-theoretic Perspective. *Fundam. Inform.*, 121(1-4):253–274, 2012.

39 Garrel Pottinger. A Type Assignment for the Strongly Normalizable λ-terms. In *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 561–577. Academic Press, 1980.

40 Jakob Rehof and Pawel Urzyczyn. The Complexity of Inhabitation with Explicit Intersection. In *Logic and Program Semantics - Essays Dedicated to Dexter Kozen on the Occasion of His 60th Birthday*, pages 256–270, 2012.

41 John C. Reynolds. Preliminary Design of the Programming Language Forsythe. Report CMU–CS–88–159, Carnegie Mellon University, june 21 1988.

42 Simona Ronchi Della Rocca and Luca Roversi. Intersection logic. In *Proc. of CSL*, volume 2142 of *Lecture Notes in Computer Science*, pages 421–428. Springer-Verlag, 2001.

43 Claude Stolze, Luigi Liquori, Furio Honsell, and Ivan Scagnetto. Towards a Logical Framework with Intersection and Union Types. In *Proc. of LFMTP*, pages 1–9, 2017.

44 Masako Takahashi. Parallel reductions in λ-calculus. *Information and computation*, 118(1):120–127, 1995.

45 Pawel Urzyczyn. The Emptiness Problem for Intersection Types. *J. Symb. Log.*, 64(3):1195–1215, 1999.

46 Steffen van Bakel. Cut-Elimination in the strict intersection type assignment system is strongly normalizing. *Notre Dame Journal of Formal Logic*, 45(1):35–63, 2004.

47 Vincent van Oostrom and Femke van Raamsdonk. Weak orthogonality implies confluence: the higher-order case. In *Proc. of LFCS*, volume 813 of *Lecture Notes in Computer Science*, pages 379–392. Springer-Verlag, 1994.

48 Betti Venneri. Intersection Types as Logical Formulae. *J. Log. Comput.*, 4(2):109–124, 1994.

49 Joe B. Wells, Allyn Dimock, Robert Muller, and Franklyn Turbak. A calculus with polymorphic and polyvariant flow types. *J. Funct. Program.*, 12(3):183–227, 2002.

50 Joe B. Wells and Christian Haack. Branching Types. In *Proc. of ESOP*, volume 2305 of *Lecture Notes in Computer Science*, pages 115–132. Springer-Verlag, 2002.

Pointers in Recursion: Exploring the Tropics

Paulin Jacobé de Naurois
CNRS, Université Paris 13, Sorbonne Paris Cité, LIPN, UMR 7030, F-93430 Villetaneuse, France
denaurois@lipn.univ-paris13.fr

Abstract

We translate the usual class of partial/primitive recursive functions to a pointer recursion framework, accessing actual input values via a pointer reading unit-cost function. These pointer recursive functions classes are proven equivalent to the usual partial/primitive recursive functions. Complexity-wise, this framework captures in a streamlined way most of the relevant sub-polynomial classes. Pointer recursion with the safe/normal tiering discipline of Bellantoni and Cook corresponds to polylogtime computation. We introduce a new, non-size increasing tiering discipline, called tropical tiering. Tropical tiering and pointer recursion, used with some of the most common recursion schemes, capture the classes logspace, logspace/polylogtime, ptime, and NC. Finally, in a fashion reminiscent of the safe recursive functions, tropical tiering is expressed directly in the syntax of the function algebras, yielding the tropical recursive function algebras.

2012 ACM Subject Classification Software and its engineering → Recursion; Theory of computation → Complexity theory and logic; Theory of computation → Complexity classes

Keywords and phrases Implicit Complexity, Recursion Theory

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.29

Funding *Paulin Jacobé de Naurois*: Work partially supported by ANR project ELICA - ANR-14-CE25-0005.

Acknowledgements I am grateful to the anonymous referees for their insightful and useful feedback.

Introduction

Characterizing complexity classes without explicit reference to the computational model used for defining these classes, and without explicit bounds on the resources allowed for the calculus, has been a long term goal of several lines of research in computer science. One rather successful such line of research is recursion theory. The foundational work here is the result of Cobham [8], who gave a characterization of polynomial time computable functions in terms of bounded recursion on notations - where, however, an explicit polynomial bound is used in the recursion scheme. Later on, Leivant [12] refined this approach with the notion of tiered recursion: explicit bounds are no longer needed in his recursion schemes. Instead, function arguments are annotated with a static, numeric denotation, a *tier*, and a tiering discipline is imposed upon the recursion scheme to enforce a polynomial time computation bound. A third important step in this line of research is the work of Bellantoni and Cook [3], whose safe recursion scheme uses only syntactical constraints akin to the use of only two tier values, to characterize, again, the class of polynomial time functions.

Cobham's approach has also later on been fruitfully extended to other, important complexity classes. Results relevant to our present work, using explicitly bounded recursion, are those of Lind [16] for logarithmic space, and Allen [1] and Clote [7] for small parallel classes.

Later on, Bellantoni and Cook's purely syntactical approach proved also useful for characterizing other complexity classes. Leivant and Marion [15, 14] used a predicative version of the safe recursion scheme to characterize alternating complexity classes, while Bloch [4], Bonfante et al [5] and Kuroda[11], gave characterizations of small, polylogtime, parallel complexity classes. An important feature of these results is that they use, either

© Paulin Jacobé de Naurois;
licensed under Creative Commons License CC-BY
4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019).
Editor: Herman Geuvers; Article No. 29; pp. 29:1–29:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

explicitly or not, a tree-recursion on the input. This tree-recursion is implicitly obtained in Bloch's work by the use of an extended set of basic functions, allowing for a dichotomy recursion on the input string, while it is made explicit in the recursion scheme in the two latter works. As a consequence, these characterizations all rely on the use of non-trivial basic functions, and non-trivial data structures. Moreover, the use of distinct basic function sets and data structures make it harder to express these characterizations in a uniform framework.

Among all these previous works on sub-polynomial complexity classes, an identification is assumed between the argument of the functions of the algebra, on one hand, and the computation input on the other hand: an alternating, logspace computation on input $\overline{x}$ is denoted by a recursive function with argument $\overline{x}$. While this seems very natural for complexity classes above linear time, it actually yields a fair amount of technical subtleties and difficulties for sub-linear complexity classes. Indeed, following Chandra et al. [6] seminal paper, sub-polynomial complexity classes need to be defined with a proper, subtler model than the one-tape Turing machine: the random access Turing machine (RATM), where computation input is accessed via a unit-cost pointer reading instruction. RATM input is thus accessed via a read-only instruction, and left untouched during the computation - a feature quite different to that of a recursive function argument. Our proposal here is to use a similar construct for reading the input in the setting of recursive functions: our functions will take as input pointers on the computation input, and one-bit pointer reading will be assumed to have unit cost. Actual computation input are thus implicit in our function algebras: the fuel of the computational machinery is only pointer arithmetics. This proposal takes inspiration partially from the Rational Bitwise Equations of [5].

Following this basic idea, we then introduce a new tiering discipline, called *tropical tiering*, to enforce a non-size increasing behavior on our recursive functions, with some inspirations taken from previous works of M. Hofmann [9, 10]. Tropical tiering induces a polynomial interpretation in the tropical ring of polynomials - the ring of polynomials over the tropical ring $\mathbb{Z} \cup \{-\infty\}$, with max and + operations - and yields a characterization of logarithmic space. Compared to the characterization of logarithmic space of Neergaard [17], our algebra does not rely on an affine upper bound on the occurrences of safe arguments in the recursion and composition schemes, which proves useful for combining this approach with others. Worth mentioning also, Bellantoni [2] provides a characterization of logspace over unary numerals.

Subsequently, the use of different, classical recursion schemes over this new tropical tiering discipline yields characterizations of other, sub-polynomial complexity classes such as polylogtime, NC, and the full polynomial time class. Following the approach of Bellantoni and Cook, we furthermore embed the tiering discipline directly in the syntax, with only finitely many different tier values - four tier values in our case, instead of only two tier values for the safe recursive functions, and provide purely syntactical characterizations of these complexity classes in a unified, simple framework. Compared to previous works, our framework uses a unique, and rather minimal set of unit-cost basic functions, computing indeed basic tasks, and a unique and also simple data structure. While the syntax of our tropical composition and recursion schemes may appear overwhelming at first sight, it has the nice feature, shared with the safe recursion functions of [3], of only adding a fine layer of syntactic sugar over the usual composition and primitive recursion schemes. Removing this sugar allows to retrieve the classical schemes. In that sense, we claim our approach to be simpler than the previous ones of [4, 5, 11].

The paper is organized as follows. Section 1 introduces the notations, and the framework of pointer recursion. Section 2 applies this framework to primitive recursion. Pointer partial/primitive recursive functions are proven to coincide with their classical counterparts

in Theorem 2. Section 3 applies this framework to safe recursion on notations. Pointer safe recursive functions are proven to coincide with polylogtime computable functions in Theorem 3. Tropical tiering is defined in Section 4. Proposition 4 establishes the tropical interpretation induced by tropical tiering. Tropical recursive functions are then introduced in Subsection 4.3. Section 5 gives a sub-algebra of the former, capturing logspace/polylogtime computable functions in Theorem 9. Finally, Section 6 explores tropical recursion with substitutions, and provides a characterization of P in Theorem 11 and of NC in Theorem 13.

1 Recursion

1.1 Notations, and Recursion on Notations

Data structures considered in our paper are finite words over a finite alphabet. For the sake of simplicity, we consider the finite, boolean alphabet $\{\mathbf{0},\mathbf{1}\}$. The set of finite words over $\{\mathbf{0},\mathbf{1}\}$ is denoted as $\{\mathbf{0},\mathbf{1}\}^*$.

Finite words over $\{\mathbf{0},\mathbf{1}\}$ are denoted with overlined variables names, as in $\overline{x}$. Single values in $\{\mathbf{0},\mathbf{1}\}$ are denoted as plain variables names, as in x. The empty word is denoted by ε, while the dot symbol "." denotes the concatenation of two words as in $a.\overline{x}$, the finite word obtained by adding an a in front of the word $\overline{x}$. Finally, finite arrays of boolean words are denoted with bold variable names, as in $\mathbf{x} = (\overline{x_1}, \cdots, \overline{x_n})$. When defining schemes, we will often omit the length of the arrays at hand, when clear from context, and use bold variable names to simplify notations. Similarly, for mutual recursion schemes, finite arrays of mutually recursive functions are denoted by a single bold function name. In this case, the *width* of this function name is the size of the array of the mutually recursive functions.

Natural numbers are identified with finite words over $\{\mathbf{0},\mathbf{1}\}$ via the usual binary encoding. Yet, in most of our function algebras, recursion is not performed on the numerical value of an integer, as in classical primitive recursion, but rather on its boolean encoding, that is, on the finite word over $\{\mathbf{0},\mathbf{1}\}$ identified with it: this approach is denoted as *recursion on notations*.

1.2 Turing Machines with Random Access

When considering sub-polynomial complexity class, classical Turing Machines often fail to provide a suitable cost model. A crucial example is the class DLOGTIME: in logarithmic time, a classical Turing machine fails to read any further than the first $k.\log(n)$ input bits. In order to provide a suitable time complexity measure for sub-polynomial complexity classes, Chandra et al [6] introduced the Turing Machine with Random Access (RATM), whose definition follows.

▶ **Definition 1** (RATM). *A Turing Machine with Random Access (RATM) is a Turing machine with no input head, one (or several) working tapes and a special* pointer *tape, of logarithmic size, over a binary alphabet. The Machine has a special* Read *state such that, when the binary number on the pointer tape is k, the transition from the Read state consists in writing the kth input symbol on the (first) working tape.*

1.3 Recursion on Pointers

In usual recursion theory, a function computes a value on its input, which is given explicitly as an argument. This, again, is the case in classical primitive recursion. While this is suitable for describing explicit computation on the input, as, for instance for single tape Turing Machines, this is not so for describing input-read-only computation models, as, for instance,

RATMs. In order to propose a suitable recursion framework for input-read-only computation, we propose the following *pointer recursion* scheme, whose underlying idea is pretty similar to that of the RATM.

As above, recursion data is given by finite, binary words, and the usual recursion on notation techniques on these recursion data apply. The difference lies in the way the actual computation input is accessed: in our framework, we distinguish two notions, the *computation input*, and the *function input*: the former denotes the input of the RATM, while the latter denotes the input in the function algebra. For classical primitive recursive functions, the two coincide, up to the encoding of integer into binary strings. In our case, we assume an explicit encoding of the former into the latter, given by the two following constructs.

Let $\overline{w} = w_1.\cdots.w_n \in \{\mathbf{0},\mathbf{1}\}^*$ be a computation input. To $\overline{w}$, we associate two constructs,

- the `Offset`: a finite word over $\{\mathbf{0},\mathbf{1}\}$, encoding in binary the length n of $\overline{w}$, and
- the `Read` construct, a 1-ary function, such that, for any binary encoding $\bar{i}$ of an integer $0 < i \leq n$, $\texttt{Read}(\bar{i}) = w_i$, and, for any other value $\overline{v}$, $\texttt{Read}(\overline{v}) = \varepsilon$.

Then, for a given *computation input* $\overline{w}$, we fix accordingly the semantics of the `Read` and `Offset` constructs as above, and a *Pointer Recursive function* over $\overline{w}$ is evaluated with sole input the `Offset`, accessing computation input bits via the `Read` construct. For instance, under these conventions, `Read(hd(Offset))` outputs the first bit of the computational input $\overline{w}$. In some sense, the two constructs depend on $\overline{w}$, and can be understood as functions on $\overline{w}$. However, in our approach, it is important to forbid $\overline{w}$ from appearing explicitly as a function argument in the syntax of the function algebras we will define, and from playing any role in the composition and recursion schemes. Since $\overline{w}$ plays no role at the syntactical level - its only role is at the semantical level- we chose to remove it completely from the syntactical definition of our functions algebras.

2 Pointers Primitive Recursion

Let us first detail our pointer recursive framework for the classical case of primitive recursion on notations.

Basic pointer functions

Basic pointer functions are the following kind of functions:

1. Functions manipulating finite words over $\{\mathbf{0},\mathbf{1}\}$. For any $a \in \{\mathbf{0},\mathbf{1}\}, \overline{x} \in \{\mathbf{0},\mathbf{1}\}^*$,

$$\begin{array}{rclrclrcl} \texttt{hd}(a.\overline{x}) & = & a & \texttt{tl}(a.\overline{x}) & = & \overline{x} & \texttt{s}_0(\overline{x}) & = & 0.\overline{x} \\ \texttt{hd}(\varepsilon) & = & \varepsilon & \texttt{tl}(\varepsilon) & = & \varepsilon & \texttt{s}_1(\overline{x}) & = & 1.\overline{x} \end{array}$$

2. Projections. For any $n \in \mathbb{N}$, $1 \leq i \leq n$,

$$\texttt{Pr}_i^n(\overline{x_1}, \cdots, \overline{x_n}) = \overline{x_i}$$

3. and, finally, the `Offset` and `Read` constructs, as defined above.

Composition

Given functions g, and $h_1, \cdots, h_n$, we define f by composition as

$$f(\mathbf{x}) = g(h_1(\mathbf{x}), \cdots, h_n(\mathbf{x})).$$

Primitive Recursion on Notations

Let $\perp$ denote non-terminating computation. Given functions h, g_0 and g_1, we define f by primitive recursion on notations as

$$\begin{array}{rcl} f(\varepsilon, \mathbf{y}) & = & h(\mathbf{y}) \\ f(\mathtt{s}_a(\overline{x}), \mathbf{y}) & = & \left\{ \begin{array}{ll} g_a(\overline{x}, f(\overline{x}, \mathbf{y}), \mathbf{y}) & \text{if } f(\overline{x}, \mathbf{y}) \neq \perp \\ \perp & \text{otherwise.} \end{array} \right. \end{array}$$

Minimization

For a function s, denote by $s^{(n)}$ its nth iterate. Then, given a function h, we define f by minimization on $\overline{x}$ as

$$\mu\overline{x}(h(\overline{x}, \mathbf{y})) = \left\{ \begin{array}{ll} \perp & \text{if } \forall t \in \mathbb{N}, \mathtt{hd}(h(\mathtt{s}_0^{(t)}(\varepsilon), \mathbf{y})) \neq \mathtt{s}_1(\varepsilon) \\ \mathtt{s}_0^{(k)}(\varepsilon) \quad \text{where } k = \min\{t \ : \mathtt{hd}(h(\mathtt{s}_0^{(t)}(\varepsilon), \mathbf{y})) = \mathtt{s}_1(\varepsilon)\} & \text{otherwise.} \end{array} \right.$$

In other words, a function f defined by minimization on h produces the shortest sequence of 0 symbols satisfying a simple condition on h , if it exists.

Let now PR_{not}^{point} be the closure of basic pointer functions under composition and primitive recursion on notations, and REC_{not}^{point} be the closure of basic pointer functions under composition, primitive recursion on notations, and minimization. Then, as expected,

▶ **Theorem 2.** *Modulo the binary encoding of natural integers, PR_{not}^{point} is the classical class of primitive recursive functions, and REC_{not}^{point} is the classical class of recursive functions.*

Proof. It is already well known that primitive recursive functions on notations are the classical primitive recursive functions, and recursive functions on notations are the classical recursive functions. Now, for one direction, it suffices to express the `Read` and `Offset` basic pointer functions as primitive recursive functions on the computation input. For the other direction, it suffices to reconstruct with pointer primitive recursion the computation input from the `Read` and `Offset` basic pointer functions. ◀

A Simple Example

Let us define the following functions.

$$\begin{array}{rcl} \mathtt{if}_\varepsilon(\varepsilon, \overline{y}, \overline{z}) & = & \overline{y} \\ \mathtt{if}_\varepsilon(\mathtt{s}_a(\overline{x}), \overline{y}, \overline{z}) & = & \overline{z} \end{array}$$

$$\begin{array}{rcl} \mathtt{RmvLastBit}(\varepsilon) & = & \varepsilon \\ \mathtt{RmvLastBit}(\mathtt{s}_a(\overline{x})) & = & \mathtt{if}_\varepsilon(\overline{x}, \varepsilon, \mathtt{s}_a(\mathtt{RmvLastBit}(\overline{x}))) \end{array}$$

Then, `Read(RmvLastBit(Offset))` reads the middle bit of the computation input, in logarithmic time. This exemplifies the purpose of recursion on pointers: this simple task cannot be performed in less than linear time by the usual primitive recursion on notations, nor by the classical Turing machine. However, switching the model definition from Turing machines to RATMs, and from primitive recursion to pointer primitive recursion, allows to perform this simple task in logarithmic time, in a straightforward way.

3 Pointer Safe Recursion

We recall the tiering discipline of Bellantoni and Cook [3]: function arguments are divided into two tiers, *normal* arguments and *safe* arguments. Notation-wise, both tiers are separated by a semicolon symbol in a block of arguments, the normal arguments being on the left, and the safe arguments on the right. We simply apply this tiering discipline to our pointer recursion framework.

Basic Pointer Safe Functions

Basic pointer safe functions are the basic pointer functions of the previous section, all their arguments being considered safe.

Safe Composition

Safe composition is somewhat similar to the previous composition scheme, with a tiering discipline, ensuring that safe arguments cannot be moved to a normal position in a function call. The reverse however is allowed.

$$f(\mathbf{x};\mathbf{y}) = g(h_1(\mathbf{x};), \cdots, h_m(\mathbf{x};); h_{m+1}(\mathbf{x};\mathbf{y}), \cdots, h_{m+n}(\mathbf{x};\mathbf{y})).$$

Calls to functions h_{m+i}, where safe arguments are used, are placed in safe position in the argument block of g. A special case of safe composition is $f(\overline{x};\overline{y}) = g(;\overline{x},\overline{y})$, where a normal argument $\overline{x}$ is used in safe position in a call. Hence, we liberally use normal arguments in safe position, when necessary.

Safe Recursion

The recursion argument is normal. The recursive call is placed in safe position, a feature that prevents nesting recursive calls exponentially.

$$\begin{aligned} f(\varepsilon,\mathbf{y};\mathbf{z}) &= h(\mathbf{y};\mathbf{z}) \\ f(a.\overline{x},\mathbf{y};\mathbf{z}) &= g_a(\overline{x},\mathbf{y}; f(\overline{x},\mathbf{y};\mathbf{z}),\mathbf{z}). \end{aligned}$$

Let now SR_{not}^{point} be the closure of the basic pointer safe functions under safe composition and safe recursion.

▶ **Theorem 3.** *SR_{not}^{point} is the class $DTIME(polylog)$ of functions computable in polylogarithmic time.*

Proof. The proof is essentially the same as for the classical result by Bellantoni and Cook [3]. Here however, it is crucial to use the RATM as computation model. Simulating a polylogtime RATM with safe recursion on pointers is very similar to simulating a polytime TM with safe recursion - instead of explicitly using the machine input as recursion data, we use the size of the input as recursion data, and access the input values via the `Read` construct, exactly as is done by the RATM model. The other direction is also similar: the tiering discipline of the safe recursion on pointers enforces a polylog bound on the size of the strings (since the initial recursion data - the `Offset` - has size logarithmic in the size n of the computation input), and thus a polylog bound on the computation time. ◀

4 Tropical Tiering

We present here another, stricter tiering discipline, that we call *tropical Tiering*. The adjective "tropical" refers to the fact that this tiering induces a polynomial interpretation in the tropical ring of polynomials. This tiering discipline takes some inspiration from Hofmann's work on non-size increasing types [9], and pure pointer programs [10]. The idea however is to use here different tools than Hofmann's to achieve a similar goal of bounding the size of the function outputs. We provide here a non-size increasing discipline via the use of tiering, and use it in the setting of pointer recursion to capture not only pure pointer programs (Hoffman's class), but rather pointer programs with pointer arithmetics, which is in essence the whole class Logspace.

Basic Pointer Functions

We add the following numerical successor basic function. Denote by $E : \mathbb{N} \to \{\mathbf{0}, \mathbf{1}\}^*$ the usual binary encoding of integers, and $D : \{\mathbf{0}, \mathbf{1}\}^* \to \mathbb{N}$ the decoding of binary strings to integers. Then,

$$\mathtt{s}(\overline{x}) = E(D(\overline{x}) + 1)$$

denotes the numerical successor on binary encodings, and, by convention, ε is the binary encoding of the integer 0.

Primitive Recursion on Values

Primitive recursion on values is the usual primitive recursion, encoded into binary strings:

$$\begin{aligned} f(\varepsilon, \mathbf{y}) &= h(\mathbf{y}) \\ f(\mathtt{s}(\overline{x}), \mathbf{y}) &= g(\overline{x}, f(\overline{x}, \mathbf{y}), \mathbf{y}). \end{aligned}$$

4.1 Tropical Tier

As usual, tiering consists in assigning function variables to different classes, called tiers. In our setting, these tiers are identified by a numerical value, called *tropical tier*, or, shortly, *tropic*. The purpose of our tropical tiers is to enforce a strict control on the increase of the size of the binary strings during computation. Tropics take values in $\mathbb{Z} \cup \{-\infty\}$. The tropic of the ith variable of a function f is denoted $T_i(f)$. The intended meaning of the tropics is to provide an upper bound on the linear growth of the function output size with respect to the corresponding input size, as per Proposition 4. Tropics are inductively defined as follows.

1. Basic pointer functions:

$$\begin{array}{rclrclrcl} T_{j \neq i}(\mathtt{Pr}_i^n) &=& -\infty & T_1(\mathtt{hd}) &=& -\infty & T_1(\mathtt{Read}) &=& -\infty \\ T_1(\mathtt{tl}) &=& -1 & & & & & & \\ T_i(\mathtt{Pr}_i^n) &=& 0 & & & & & & \\ T_1(\mathtt{s}_0) &=& 1 & T_1(\mathtt{s}_1) &=& 1 & T_1(\mathtt{s}) &=& 1 \end{array}$$

2. Composition:

$$T_t(f) = \max_i \{T_i(g) + T_t(h_i)\}.$$

3. Primitive recursion on notations. Two cases arise:

- $T_2(g_0) \le 0$ and $T_2(g_1) \le 0$. In that case, we set
  a. $T_1(f) = \max\ \{T_1(g_0), T_1(g_1), T_2(g_0), T_2(g_1)\}$, and,
  b. for all $t > 1$,
  $T_t(f) = \max\{T_{t+1}(g_0), T_{t+1}(g_1), T_{t-1}(h), T_2(g_0), T_2(g_1)\}$.
- the previous case above does not hold, $T_2(g_0) \le 1$, and $T_2(g_1) \le 1$. In that case, we also require that $T_1(g_0) \le 0$, $T_1(g_1) \le 0$, and, for all $t \ge 2$, $T_t(g_0) = T_t(g_1) = T_{t-2}(h) = -\infty$. Then, we set $T_1(f) = \max\{T_1(g_0), T_1(g_1), T_2(g_0) - 1, T_2(g_1) - 1, c_h\}$, where c_h is a constant for h given in Proposition 4 below, and, for $t > 1$, $T_t(f) = -\infty$.

Other cases than the two above do not enjoy tropical tiering.

4. Primitive recursion on values. Only one case arises:
- $T_2(g) \le 0$. In that case, we set
  a. $T_1(f) = \max\ \{T_1(g), T_2(g)\}$, and,
  b. for all $t > 1$, $T_t(f) = \max\{T_{t+1}(g), T_{t-1}(h), T_2(g)\}$.

Again, other cases than the one above do not enjoy tropical tiering.

Furthermore, when using tropical tiering, we use mutual recursion schemes. For $\mathbf{f} = (f_1, \cdots, f_n)$, mutual primitive recursion (on values) is classically defined as follows,

$$\begin{array}{rcl} \mathbf{f}(\varepsilon, \mathbf{y}) & = & \mathbf{h}(\mathbf{y}) \\ \mathbf{f}(\mathbf{s}(\overline{x}), \mathbf{y}) & = & \begin{cases} \mathbf{g}(\overline{x}, \mathbf{f}(\overline{x}, \mathbf{y}), \mathbf{y}) & \text{if } \forall i\ (f_i(\overline{x}, \mathbf{y}) \neq \perp) \\ \perp & \text{otherwise.} \end{cases} \end{array}$$

and similarly for mutual primitive recursion on notations. Tropical tiering is then extended to mutual primitive recursion in a straightforward manner.

We define the set of L-primitive pointer recursive functions as the closure of the basic pointer functions of Sections 2 and 4 under composition, (mutual) primitive recursion on notations and (mutual) primitive recursion on values, with tropical tiering.

4.2 Tropical Interpretation

Tropical tiering induces a non-size increasing discipline. More formally,

▶ **Proposition 4.** *The tropical tiering of a L-primitive recursive function f induces a polynomial interpretation of f on the tropical ring of polynomials, as follows.*

For any L-primitive recursive function f with n arguments, there exists a computable constant $c_f \ge 0$ such that

$$|f(\overline{x_1}, \cdots, \overline{x_n})| \le \max_t\{T_t(f) + |\overline{x_t}|, c_f\}.$$

Proof. The proof is given for non-mutual recursion schemes, by induction on the definition tree. Mutual recursion schemes follow the same pattern.

1. For basic pointer functions, the result holds immediately.

2. Let f be defined by composition, and assume that the result holds for the functions g, $h_1, \cdots, h_n$. Then, for any $i = 1, \cdots, n$, $|h_i(\mathbf{x})| \le \max_t\{T_t(h_i) + |\overline{x_t}|, c_{h_i}\}$. Moreover, there exists by induction c_g such that $|g(h_1(\mathbf{x}), \cdots, h_n(\mathbf{x}))| \le \max_i\{T_i(g) + |h_i(\mathbf{x})|, c_g\}$. Composing the inequalities above yields $|g(h_1(\mathbf{x}), \cdots, h_n(\mathbf{x}))| \le \max_i\{T_i(g) + \max_t\{T_t(h_i) + |\overline{x_i}|, c_{h_i}\}, c_g\} = \max_t\{T_t(f) + |\overline{x_t}|, \max_i\{c_{f_i}, c_g\}\}$.

3. Let f be defined by primitive recursion on notations, and assume that the first case holds. Let $f(a.\overline{x}, \mathbf{y}) = g_a(\overline{x}, f(\overline{x}, \mathbf{y}), \mathbf{y})$, for $a \in \{0, 1\}$, and assume $T_2(g_0) \le 0$ and $T_2(g_1) \le 0$. We apply the tropical interpretation on g, and we show by induction the result for f on the length of $a.\overline{x}$.

a. If $\max_{\overline{x},f(\overline{x},\mathbf{y}),t}\{|\overline{x}| + T_1(g_a), |f(\overline{x},\mathbf{y})| + T_2(g_a), |\overline{y_t}| + T_{t+2}(g_a), c_{g_a}\} = |\overline{x}| + T_1(g_a)$: $|f(a.\overline{x},\mathbf{y})| \leq |\overline{x}| + T_1(g_a) \leq |\overline{x}| + T_1(f)$, and the result holds.

b. If $\max_{\overline{x},f(\overline{x},\mathbf{y}),t}\{|\overline{x}|+T_1(g_a), |f(\overline{x},\mathbf{y})|+T_2(g_a), |\overline{y_t}|+T_{t+2}(g_a), c_{g_a}\} = |f(\overline{x},\mathbf{y})|+T_2(g_a)$: Since $T_2(g_a) \leq 0$, $|f(a.\overline{x},\mathbf{y})| \leq |f(\overline{x},\mathbf{y})|$, and the induction hypothesis applies.

c. If $\max_{\overline{x},f(\overline{x},\mathbf{y}),t}\{|\overline{x}| + T_1(g_a), |f(\overline{x},\mathbf{y})| + T_2(g_a), |\overline{y_t}| + T_{t+2}(g_a), c_{g_a}\} = |\overline{y_t}| + T_{t+2}(g_a)$ for some t: the result applies immediately by structural induction on g_a.

d. If $\max_{\overline{x},f(\overline{x},\mathbf{y}),t}\{|\overline{x}| + T_1(g_a), |f(\overline{x},\mathbf{y})| + T_2(g_a), |\overline{y_t}| + T_{t+2}(g_a), c_{g_a}\} = c_{g_a}$, the result holds immediately.

e. The base case $f(\epsilon,\mathbf{y})$ is immediate.

4. Let f be defined by primitive recursion on notations, and assume now that the second of the two corresponding cases holds. Let $f(a.\overline{x},\mathbf{y}) = g_a(\overline{x}, f(\overline{x},\mathbf{y}),\mathbf{y})$, for $a \in \{0,1\}$. Since the first case does not hold, $T_2(g_0) = 1$ or $T_2(g_1) = 1$: assume that $T_2(g_0) = 1$ (the other case being symmetric). Assume also that, $T_1(g_0) \leq 0$ and $T_1(g_1) \leq 0$, and for all $t \geq 2$, $T_t(g_0) = T_t(g_1) = T_{t-2}(h) = -\infty$. Then, we set $T_1(f) = \max\{0, c_h\}$. We apply the tropical interpretation on g, and prove by induction on the length of $a.\overline{x}$ that $|f(a.\overline{x},\mathbf{y})| \leq |a.\overline{x}| + \max\{c_{g_1}, c_{g_2}, c_h\}$.

a. If $\max_{t\geq 2}\{|\overline{x}| + T_1(g_a), |f(\overline{x},\mathbf{y})| + T_2(g_a), |\overline{y_t}| + T_t(g_a), c_{g_a}\} = |\overline{x}| + T_1(g_a)$. Since $T_1(g_a) \leq 0$ and $T_1(f) \geq 0$, $|f(a.\overline{x},\mathbf{y})| \leq |\overline{x}| \leq T_1(f) + |\overline{x}|$, and the result holds.

b. If $\max_{t\geq 2}\{|\overline{x}| + T_1(g_a), |f(\overline{x},\mathbf{y})| + T_2(g_a), |\overline{y_t}| + T_t(g_a), c_{g_a}\} = |f(\overline{x})| + T_2(g_a)$. Since $T_2(g_a) \leq 1$, $|f(a.\overline{x})| \leq 1 + |f(\overline{x})|$, and the induction hypothesis allows to conclude.

c. If $\max_{t\geq 2}\{|\overline{x}| + T_1(g_a), |f(\overline{x},\mathbf{y})| + T_2(g_a), |\overline{y_t}| + T_t(g_a), c_{g_a}\} = c_{g_a}$, the result holds immediately.

d. The case $\max_{t\geq 2}\{|\overline{x}| + T_1(g_a), |f(\overline{x},\mathbf{y})| + T_2(g_a), |\overline{y_t}| + T_t(g_a), c_{g_a}\} = |\overline{y_t}| + T_t(g_a)$ is impossible since $T_t(g_a) = -\infty$ for $t \geq 2$.

e. The base case $f(\epsilon,\mathbf{y})$ is immediate.

5. Let now assume f is define by primitive recursion on values. Then, the only possible case is similar to the first case of primitive recursion on notation.

The proof by induction above emphasizes the critical difference between recursion on notation and recursion on values: the second case of the safe recursion on notations correspond to the linear, non-size increasing scanning of the input, as in, for instance,

$$f(a.\overline{x}) = s_a(f(\overline{x})).$$

This, of course, is only possible in recursion on notation, where the height of the recursive calls stack is precisely the length of the scanned input. Recursion on values fails to perform this linear scanning, since, for a given recursive argument $\overline{x}$, the number of recursive calls is then exponential in $|\overline{x}|$. ◀

Proposition 4 proves that the tropical tiering of a function yields actually a tropical polynomial interpretation for the function symbols: The right hand side of the Lemma inequality is indeed a tropical interpretation. Moreover, this interpretation is directly given by the syntax.

Furthermore, the proof also highlights why we use mutual recursion schemes instead of more simple, non-mutual ones: non-size increasing discipline forbids the use of multiplicative constants in the size of the strings. So, in order to capture a computational space of size $k.\log(n)$, we need to use k binary strings of length $\log(n)$, defined by mutual recursion.

▶ **Corollary 5.** *L-primitive pointer recursive functions are computable in logarithmic space.*

Proof. Proposition 4 ensures that the size of all binary strings is logarithmically bounded. A structural induction on the definition of f yields the result. The only critical case is that of a recursive construct. When evaluating a recursive construct, one needs simply to store all non-recursive arguments (the $\overline{y_i}$'s) in a shared memory, keep a shared counter for keeping track of the recursive argument $\overline{x}$, and use a simple `while` loop to compute successively all intermediate recursive calls leading to $f(\overline{x}, \mathbf{y})$. All these shared values have logarithmic size. The induction hypothesis ensures then that, at each step in the `while` loop, all computations take logarithmic space. The two other cases, composition and basic functions, are straightforward. ◀

In the following section, we prove the converse: logarithmic space functions can be computed by a sub-algebra of the L-primitive pointer recursive functions.

4.3 Tropical Recursion

In this section we restrict our tropical tiering approach to only four possible tier values: 1, 0, -1 and $-\infty$. While doing so, we still retain the same expressiveness. The rules for tiering are adapted accordingly. More importantly, the use of only four tier values allows us to denote these tropics directly in the syntax, in an approach similar to that of Bellantoni and Cook, by adding purely syntactical features to the composition and primitive recursion schemes. Let us take as separator symbol the following $\wr$ symbol, with leftmost variables having the highest tier. As with safe recursive functions, we allow the use of a high tier variable in a low tier position, as in, for instance,

$$f(\overline{x} \wr \overline{y} \wr \overline{z} \wr \overline{t}) = g(\ \wr \overline{y} \wr \overline{x}, \overline{z} \wr \overline{t}).$$

Our tropical recursive functions are then as follows.

Basic tropical pointer functions

Basic tropical pointer functions are the following.

$$\begin{array}{rclrcl}
\mathtt{hd}(\ \wr\ \wr\ \wr a.\overline{x}) & = & a & \mathtt{tl}(\ \wr\ \wr a.\overline{x} \wr\ ) & = & \overline{x} \\
\mathtt{hd}(\ \wr\ \wr\ \wr \varepsilon) & = & \varepsilon & \mathtt{tl}(\ \wr\ \wr \varepsilon \wr\ ) & = & \varepsilon \\
\mathtt{s}_0(\overline{x} \wr\ \wr\ \wr\ ) & = & 0.\overline{x} & \mathtt{s}_1(\overline{x} \wr\ \wr\ \wr\ ) & = & 1.\overline{x} \\
\mathtt{s}(\overline{x} \wr\ \wr\ \wr\ ) & = & E(D(\overline{x})+1) & \mathtt{Read}(\ \wr\ \wr\ \wr \overline{x}) & = & a \in \{0,1\}
\end{array}$$

$$\mathtt{Pr}_i^n(\ \wr \overline{x_i} \wr\ \wr \overline{x_1}, \cdots, \overline{x_{i-1}}, \overline{x_{i+1}}, \cdots, \overline{x_n}) = \overline{x_i}$$

Tropical composition

Define $\mathbf{t} = \mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3, \mathbf{t}_4$. The tropical composition scheme is then

$$\begin{array}{rcl}
f(\mathbf{x} \wr \mathbf{y} \wr \mathbf{z} \wr \mathbf{t}) & = & g(h_1(\ \wr \mathbf{x} \wr \mathbf{y} \wr \mathbf{t}), \cdots, h_a(\ \wr \mathbf{x} \wr \mathbf{y} \wr \mathbf{t}) \wr \\
 & & h_{a+1}(\mathbf{x} \wr \mathbf{y} \wr \mathbf{z} \wr \mathbf{t}), \cdots, h_b(\mathbf{x} \wr \mathbf{y} \wr \mathbf{z} \wr \mathbf{t}) \wr \\
 & & h_{b+1}(\mathbf{y} \wr \mathbf{z} \wr\ \wr \mathbf{t}), \cdots, h_c(\mathbf{y} \wr \mathbf{z} \wr\ \wr \mathbf{t}) \wr \\
 & & h_{c+1}(\mathbf{t}_1 \wr \mathbf{t}_2 \wr \mathbf{t}_3 \wr \mathbf{t}_4), \cdots, h_d(\mathbf{t}_1 \wr \mathbf{t}_2 \wr \mathbf{t}_3 \wr \mathbf{t}_4))
\end{array}$$

Tropical Recursion on Notations – case 1

$$\begin{array}{rcl}
\mathbf{f}(\mathbf{x} \wr \varepsilon, \mathbf{y} \wr \mathbf{z} \wr \mathbf{t}) & = & \mathbf{h}(\mathbf{x} \wr \mathbf{y} \wr \mathbf{z} \wr \mathbf{t}) \\
\mathbf{f}(\mathbf{x} \wr \mathtt{s}_a(\overline{r} \wr\ \wr\ \wr\ ), \mathbf{y} \wr \mathbf{z} \wr \mathbf{t}) & = & \mathbf{g}_a(\mathbf{x} \wr \overline{r}, \mathbf{f}(\mathbf{x} \wr \overline{r}, \mathbf{y} \wr \mathbf{z} \wr \mathbf{t}), \mathbf{y} \wr \mathbf{z} \wr \mathbf{t})
\end{array}$$

Tropical Recursion on Notations – case 2 (Linear scanning)

$$\begin{aligned}
\mathbf{f}(\ \wr\ \varepsilon\ \wr\ \ \wr\ \mathbf{t}) &= \varepsilon \\
\mathbf{f}(\ \wr\ \mathbf{s}_a(\overline{r}\ \wr\ \ \wr\ \ \wr\ )\ \wr\ \ \wr\ \mathbf{t}) &= \mathbf{g}_a(\mathbf{f}(\ \wr\ \overline{r}\ \wr\ \ \wr\ \mathbf{t})\ \wr\ \overline{r}\ \wr\ \ \wr\ \mathbf{t})
\end{aligned}$$

Tropical Recursion on Values

$$\begin{aligned}
\mathbf{f}(\mathbf{x} \wr \varepsilon, \mathbf{y} \wr \mathbf{z} \wr \mathbf{t}) &= \mathbf{h}(\mathbf{x} \wr \mathbf{y} \wr \mathbf{z} \wr \mathbf{t}) \\
\mathbf{f}(\mathbf{x} \wr \mathbf{s}(\overline{r}\ \wr\ \ \wr\ \ \wr\ ), \mathbf{y} \wr \mathbf{z} \wr \mathbf{t}) &= \mathbf{g}(\mathbf{x} \wr \overline{r}, \mathbf{f}(\mathbf{x} \wr \overline{r}, \mathbf{y} \wr \mathbf{z} \wr \mathbf{t}), \mathbf{y} \wr \mathbf{z} \wr \mathbf{t})
\end{aligned}$$

As above, we use the mutual version of these recursion schemes, with the same tiering discipline. Note that, unlike previous characterizations of sub-polynomial complexity classes [4, 5, 11], our tropical composition and recursion schemes are only syntactical refinements of the usual composition and primitive recursion schemes - removing the syntactical sugar yields indeed the classical schemes.

▶ **Definition 6** (L-tropical functions). *The class of L-tropical functions is the closure of our basic tropical pointer functions, under tropical composition, tropical mutual recursion on notations, and tropical mutual recursion on values.*

The restriction of only four tier values suffices to capture the computational power of RATMs. More precisely,

▶ **Theorem 7.** *The class of L-tropical functions is the class of functions computable in logarithmic space, with logarithmic size output.*

Proof. L-tropical functions are L- primitive pointer recursive functions with tropics 1, 0, -1 and $-\infty$. Following Corollary 5, they are computable in logspace. The converse follows from the simulation of a logarithmic space RATM, as follows.

Some Assumptions on the RATM being simulated. Let f be a function computable in deterministic space $k\log(n)$, with output of size $k\log(n)$, computed by a RATM M. We assume the following.

- The machine M uses one pointer tape, of size $\lceil\log(n+1)\rceil$, and exactly one computation tape.
- For every input $\overline{x}$ of length n, the machine uses exactly $k.\lceil\log(n+1)\rceil$ cells on the computation tape.
- At the start of the computation, the computation tape is as follows.
  1. The computation tape is on a cell containing the **0** symbol, followed by $k.\lceil\log(n+1)\rceil - 1$ **0** cells on the right.
  2. The cells on the left of the computation head, and the cells on the right of the $k.\lceil\log(n+1)\rceil$ **0** symbols, contain only blank symbols.
- Moreover, during the computation, the following holds.
  1. The computation head never goes on any cell on the left of its initial position.
  2. The machine never writes a blank symbol.
- The same assumptions are made for the pointer tape.

It is easy to check that these assumptions are benign. They enable us to ignore the blank symbol in the simulation, and have a strict correspondence between the binary symbols of the RATM and those of the L-tropical algebra.

Encoding the machine configurations. Assume the machine M works in space $k\lceil\log(n{+}1)\rceil$. A configuration of M is then encoded by $2k+3$ binary strings of length less than $\lceil\log(n+1)\rceil$:

1. one string, of constant length, encodes the machine state,
2. one string, of length $\lceil\log(n+1)\rceil$, encodes the pointer tape,
3. one string, of length $\lceil\log(n+1)\rceil$, encodes the head of the pointer tape. It contains **0** symbols everywhere, but on the position of the head (where it contains a **1**).
4. k strings, of length $\lceil\log(n+1)\rceil$, encode the content of the work tape, and
5. k strings, of length $\lceil\log(n+1)\rceil$, encode the position of the work tape head, with (as for the pointer tape) **0** everywhere but on the position of the head.

Reading and Updating a configuration. Linear scanning of the recursive argument in tropical recursion, corresponding to case 2 of the definition of tropical recursion on notations, is used to read and to update the encoding of the configuration. In order to do so,

1. we encode booleans `false` and `true` with $\mathtt{s}_0(\varepsilon \wr\ \wr\ \wr\ )$ and $\mathtt{s}_1(\varepsilon \wr\ \wr\ \wr\ )$ respectively. We define the following `match` construct

$$\begin{array}{lll} \mathtt{match} & \overline{x} & \mathtt{with} \\ & \mid & \mathtt{s}_0(\overline{r} \wr\ \wr\ \wr\ ) \to A \\ & \mid & \mathtt{s}_1(\overline{r} \wr\ \wr\ \wr\ ) \to B \\ & \mid & \varepsilon \to C \end{array}$$

   as the following degenerate tropical recursion on notations.

$$\begin{array}{rcl} \mathtt{match}(\ \wr\ \mathtt{s}_0(\overline{r} \wr\ \wr\ \wr\ ), \overline{a}, \overline{b}, \overline{c} \wr\ \wr\ ) & = & \overline{a} \\ \mathtt{match}(\ \wr\ \mathtt{s}_1(\overline{r} \wr\ \wr\ \wr\ ), \overline{a}, \overline{b}, \overline{c} \wr\ \wr\ ) & = & \overline{b} \\ \mathtt{match}(\ \wr\ \varepsilon, \overline{a}, \overline{b}, \overline{c} \wr\ \wr\ ) & = & \overline{c} \end{array}$$

   Then, `if then esle`, and `AND` and `OR` boolean functions are obtained by trivial applications of the `match` construct above. We also use a function `isempty`, for testing if a string equals ε.

2. we define the following function, which adds one-bit in first position.

$$\begin{array}{llll} \mathtt{1BC}(\overline{y} \wr \overline{x} \wr\ \wr\ ) = \mathtt{match} & \overline{x} & \mathtt{with} \\ & \mid & \mathtt{s}_0(\overline{t} \wr\ \wr\ \wr\ ) \to \mathtt{s}_0(\overline{y} \wr\ \wr\ \wr\ ) \\ & \mid & \mathtt{s}_1(\overline{t} \wr\ \wr\ \wr\ ) \to \mathtt{s}_1(\overline{y} \wr\ \wr\ \wr\ ) \\ & \mid & \varepsilon \to \overline{y} \end{array}$$

   For notational purposes we sometimes use $\mathtt{hd}(\ \wr\ \wr\ \wr \overline{x}).\overline{y}$ instead.

3. we define the following tail extraction, extracting the tail of a string, for a given prefix length.

$$\begin{array}{rcl} \mathtt{Te}(\ \wr\ \mathtt{s}_a(\overline{x} \wr\ \wr\ \wr\ ) \wr \overline{e} \wr\ ) & = & \mathtt{tl}(\ \wr\ \wr\ \mathtt{Te}(\ \wr \overline{x} \wr \overline{e} \wr\ ) \wr\ ) \\ \mathtt{Te}(\ \wr\ \varepsilon \wr \overline{e} \wr\ ) & = & \overline{e} \end{array}$$

4. we define the following bit extraction, extracting one bit of a string, for a given prefix length.

$$\mathtt{Be}(\ \wr\ \wr\ \wr \overline{x}, \overline{e}) = \mathtt{hd}(\ \wr\ \wr\ \wr\ \mathtt{Te}(\ \wr \overline{x} \wr \overline{e} \wr\ ))$$

5. we define the following head extraction, extracting the head of a string, for a given prefix length.
$$\texttt{He}(\wr\, \mathtt{s}_a(\wr\ \wr\ \wr\, \overline{x})\ \wr\ \wr\, \overline{e}) \;=\; \texttt{Be}(\wr\ \wr\ \wr\, \mathtt{s}_a(\wr\ \wr\ \wr\, \overline{x}), e).\texttt{He}(\wr\, \overline{x}\ \wr\ \wr\, \overline{e})$$
$$\texttt{He}(\wr\, \varepsilon\ \wr\ \wr\, \overline{e}) \;=\; \varepsilon$$
6. we define the prefix length computation, which extracts the initial subsequence of **0** only symbols, followed by the first **1**. This function is used for computing the prefix length corresponding to the position of the head in our encoding of the tapes of the RATM.
$$\texttt{Prefix}(\wr\, \varepsilon\ \wr\ \wr\,) \;=\; \varepsilon$$
$$\texttt{Prefix}(\wr\, \mathtt{s}_0(\wr\ \wr\ \wr\, \overline{x})\ \wr\ \wr\,) \;=\; \mathtt{s}_0(\texttt{Prefix}(\wr\, \overline{x}\ \wr\ \wr\,)\ \wr\ \wr\ \wr\,)$$
$$\texttt{Prefix}(\wr\, \mathtt{s}_1(\wr\ \wr\ \wr\, \overline{x})\ \wr\ \wr\,) \;=\; \mathtt{s}_1(\varepsilon\ \wr\ \wr\ \wr\,)$$
7. we define a predicate for comparing string lengths
$$\texttt{SameLength}(\wr\, \overline{x}, \overline{y}\ \wr\ \wr\,) =$$
$$\texttt{AND}(\wr\, \texttt{isempty}(\wr\, \texttt{Te}(\wr\, \overline{x}\ \wr\ \wr\, \overline{y})\ \wr\ \wr\,), \texttt{isempty}(\wr\, \texttt{Te}(\wr\, \overline{y}\ \wr\ \wr\, \overline{x})\ \wr\ \wr\,)\ \wr\ \wr\,).$$
8. we define the one bit replacement function: Replacing exactly one bit in a string $\overline{e}$ by the first bit of $\overline{b}$, for a given prefix length $\overline{x}$.
$$\begin{aligned}\texttt{Cb}(\wr\, \mathtt{s}_a(\overline{x}\ \wr\ \wr\ \wr\,)\ \wr\ \wr\, \overline{y}, \overline{e}, \overline{b}) \;=\;& \\ &\texttt{if SameLength}(\wr\, \mathtt{s}_a(\overline{x}\ \wr\ \wr\ \wr\,), \overline{y}\ \wr\ \wr\,) \\ &\quad \texttt{then hd}(\wr\ \wr\ \wr\, \overline{b}).\texttt{Cb}(\wr\, \overline{x}\ \wr\ \wr\, \overline{y}, \overline{e}, \overline{b}) \\ \texttt{else Be}(\wr\ \wr\ \wr\, \texttt{Te}(\wr\, \mathtt{s}_a(\overline{x}\ \wr\ \wr\ \wr\,)\ \wr\, \overline{e}\ \wr\,), \overline{e}).&\texttt{Cb}(\wr\, \overline{x}\ \wr\ \wr\, \overline{y}, \overline{e}, \overline{b}) \\ \texttt{Cb}(\wr\, \varepsilon\ \wr\ \wr\, \overline{y}, \overline{e}, \overline{b}) \;=\;& \varepsilon\end{aligned}$$
and
$$\texttt{ChBit}(\wr\, \overline{s}\ \wr\ \wr\, \overline{x}, \overline{e}, \overline{b}) = \texttt{Cb}(\wr\, \overline{s}\ \wr\ \wr\, \texttt{Te}(\wr\, \overline{x}\ \wr\ \wr\, \overline{e}), \overline{e}, \overline{b})$$
for any $\overline{s}$ with $|\overline{s}| = |\overline{e}|$.

With all these simple bricks, and especially with the in-place one-bit replacement, one is then able to read a configuration, and to update it, with L-tropical functions. None of these L-tropical functions uses recursion on values.

Computing the Transition map of the Machine. Given the functions above, the transition map **Next** of the machine is then computed by a simple L-tropical function of width $(2k+3)$: For a recursive argument $\overline{s}$ of size $\lceil\log(n+1)\rceil$, $\mathbf{Next}(\wr\, \overline{s}, \mathbf{c}\ \wr\ \wr\,)$ computes the configuration reached from $\mathbf{c}$ in one transition step.

The `Prefix` function above computes the prefix corresponding the position of the head of the pointer and of the computation tapes in our encoding. Used in conjunction with the boolean constructs on the k strings encoding the computation tape, and in conjunction with the bit extraction function `Be` above, it allows to read the current symbol on the computation tape, and on the pointer tape, of the encoding of the RATM. Updating these two symbols is performed with the `ChBit` in-place one-bit replacement function.

Similarly, moving the heads of these two tapes can easily be performed with this `ChBit`, in conjunction with the `tl` and $\mathtt{s}_1$ basic tropical functions.

Let us now describe how we can read and update the machine state: This machine state is encoded in binary by a string of length $\lceil\log(S+1)\rceil$, where S is the number of the states of M. The length of this string is fixed, and does not depend on the input. Therefore, we

can safely assume that we have a fixed decision tree of depth $\lceil \log(S+1) \rceil$, for reading each bit of this string. The leaves of this decision tree are in one-to-one correspondence with the states of M. This decision tree can moreover be encoded with basic tropical functions and tropical composition only. Similarly, overwriting the machine state can be done with basic tropical functions and tropical composition only.

Finally, when in an input reading state, the input tape symbol is obtained simply by using the basic tropical function `Read`, with the pointer tape as argument.

The transition map **Next** of the RATM is then obtained by a boolean composition of the above functions. Similarly, computing an encoding of the initial configuration, and reading a final configuration, is simple.

Simulating the RATM. The simulation of the RATM is then obtained by iterating its transition map **Next** a suitable number of times. The time upper bound is here obtained by nesting k tropical recursive functions on values: on an input of size $\lceil \log(n+1) \rceil$, the unfolding of these recursive calls takes time n^k. At each recursive step, this function needs to apply the transition map. The transition map having width $(2k+3)$, we use here a mutual recursion scheme, of width $(2k+3)$. Again, for a recursive argument $\overline{s}$ of size $\lceil \log(n+1) \rceil$, we define

$$\begin{aligned}
\mathbf{Step}_1(\wr\, \varepsilon, \overline{s}, \mathbf{c} \wr\ \wr) &= \mathbf{c} \\
\mathbf{Step}_1(\wr\, \mathbf{s}(\overline{t} \wr\ \wr\ \wr), \overline{s}, \mathbf{c} \wr\ \wr) &= \mathbf{Next}(\wr\, \overline{s}, \mathbf{Step}_1(\wr\, \overline{t}, \overline{n}, \mathbf{c} \wr\ \wr) \wr\ \wr) \\
\mathbf{Step}_2(\wr\, \varepsilon, \overline{s}, \mathbf{c} \wr\ \wr) &= \mathbf{c} \\
\mathbf{Step}_2(\wr\, \mathbf{s}(\overline{t} \wr\ \wr\ \wr), \overline{s}, \mathbf{c} \wr\ \wr) &= \mathbf{Step}_1(\wr\, \overline{s}, \overline{s}, \mathbf{Step}_2(\wr\, \overline{t}, \overline{s}, \mathbf{c} \wr\ \wr) \wr\ \wr) \\
&\ \vdots \\
\mathbf{Step}_k(\wr\, \varepsilon, \overline{s}, \mathbf{c} \wr\ \wr) &= \mathbf{c} \\
\mathbf{Step}_k(\wr\, \mathbf{s}(\overline{t} \wr\ \wr\ \wr), \overline{s}, \mathbf{c} \wr\ \wr) &= \mathbf{Step}_{k-1}(\wr\, \overline{s}, \overline{s}, \mathbf{Step}_k(\wr\, \overline{t}, \overline{s}, \mathbf{c} \wr\ \wr) \wr\ \wr).
\end{aligned}$$

Replacing $\overline{s}$ by the `Offset` in the above gives the correct bounds.

Finally, one simply needs to use simple L-tropical functions for computing the initial configuration, and reading the final configuration. ◀

5 Logarithmic Space, Polylogarithmic Time

The polynomial time bound in Theorem 7 relies on the use of tropical recursion on values, for clocking the simulation of the RATM. Restricting the algebra to tropical recursion on notations only yields a straightforward time bound restriction, as follows.

▶ **Definition 8** (LP-tropical functions). *The class of LP-tropical functions is the closure of our basic tropical pointer functions, under tropical composition and tropical mutual recursion on notations.*

▶ **Theorem 9.** *The class of LP-tropical functions is the class of functions computable in logarithmic space, polylogarithmic time, with logarithmic size output.*

Proof. Mutual recursion on notations, with recursive arguments of logarithmic size, are computable in polylogarithmic time, following similar arguments as in the proof of Theorem 7. The converse follows from the simulation in the proof of Theorem 7 above, where mutual recursion on values for the functions $\texttt{Step}_i$ is replaced by mutual recursion on notations. ◀

6 Alternation

In this section we extend the approach of Leivant and Marion [13] to our setting. Let us define a similar tropical recursion on notations with substitutions. Note that the tropical tiering discipline prevents using substitutions in case 2 of the tropical recursion on notations. Substitutions are therefore only defined for case 1 of this recursion scheme, and only in non-size increasing position.

Tropical Recursion with substitutions on Notations

Given functions h, g_0, g_1, k_1 and k_2,

$$\begin{aligned}
\mathbf{f}(\mathbf{x} \wr \varepsilon, \mathbf{u}, \mathbf{y} \wr \mathbf{z} \wr \mathbf{t}) &= \mathbf{h}(\mathbf{x} \wr \mathbf{u}, \mathbf{y} \wr \mathbf{z} \wr \mathbf{t}) \\
\mathbf{f}(\mathbf{x} \wr \mathsf{s}_a(\overline{r} \wr \ \wr \ \wr\ ), \mathbf{u}, \mathbf{y} \wr \mathbf{z} \wr \mathbf{t}) &= \mathbf{g}_a(\mathbf{x} \wr \overline{r}, \mathbf{f}(\mathbf{x} \wr \overline{r}, k_1(\ \wr \mathbf{u} \wr \ \wr\ ), \mathbf{y} \wr \mathbf{z} \wr \mathbf{t}), \\
&\quad \mathbf{f}(\mathbf{x} \wr \overline{r}, k_2(\ \wr \mathbf{u} \wr \ \wr\ ), \mathbf{y} \wr \mathbf{z} \wr \mathbf{t}), \mathbf{u}, \mathbf{y} \wr \mathbf{z} \wr \mathbf{t}) \ .
\end{aligned}$$

Tropical Recursion with substitutions on Values

Given functions h, g, k_1 and k_2,

$$\begin{aligned}
\mathbf{f}(\mathbf{x} \wr \varepsilon, \mathbf{u}, \mathbf{y} \wr \mathbf{z} \wr \mathbf{t}) &= \mathbf{h}(\mathbf{x} \wr \mathbf{u}, \mathbf{y} \wr \mathbf{z} \wr \mathbf{t}) \\
\mathbf{f}(\mathbf{x} \wr \mathbf{s}(\overline{r} \wr \ \wr \ \wr\ ), \mathbf{u}, \mathbf{y} \wr \mathbf{z} \wr \mathbf{t}) &= \mathbf{g}(\mathbf{x} \wr \overline{r}, f(\mathbf{x} \wr \overline{r}, k_1(\ \wr \mathbf{u} \wr \ \wr\ ), \mathbf{y} \wr \mathbf{z} \wr \mathbf{t}), \\
&\quad \mathbf{f}(\mathbf{x} \wr \overline{r}, k_2(\ \wr \mathbf{u} \wr \ \wr\ ), \mathbf{y} \wr \mathbf{z} \wr \mathbf{t}), \mathbf{u}, \mathbf{y} \wr \mathbf{z} \wr \mathbf{t}) \ .
\end{aligned}$$

Again, as above, we assume these recursion schemes to be mutual. Tropical Recursion with substitutions allows two recursive calls, where one parameter ($\mathbf{u}$) is modified by the so-called substitution function k_1 or k_2. This allows to denote branching, or parallel computations, where a given configuration being accepting or rejecting depends on two distinct transition steps. This is precisely the definition of an alternating Turing machine, whose relation with parallel computation is well documented [6, 18].

▶ **Definition 10** (P-tropical functions). *The class of P-tropical functions is the closure of our basic tropical pointer functions, under tropical composition, tropical recursion on notations and on values, and tropical recursion with substitutions on notations and on values.*

▶ **Theorem 11.** *The class of P-tropical functions with binary output is the class P.*

Proof. The result follows from *Alogspace* $= P$ [6], and Theorem 7. Substitutions in the tropical recursion scheme on notations amounts to alternation. Restriction to decision classes instead of function classes comes from the use of alternating Turing machines, which compute only decision problems.

Let us first see how to simulate a logspace alternating machine with P-tropical functions. Recall the notations and functions of the proof of Theorem 7. Since we now need to simulate a non-deterministic, alternating machine, we assume without loss of generality that we now have two kinds of machine states:

- non-deterministic universal
- non-deterministic existential

and that non-deterministic transitions have at most two branches. Therefore, we also assume that we have one predicate that determines the kind of a state in a configuration $\mathbf{c}$: `IsUniversal`$(\ \wr \overline{s}, \mathbf{c} \wr \ \wr\ )$. This predicate is assumed to output `false` or `true`.

We also assume that we have two transition maps, $\mathtt{Next}_0(\wr \overline{s}, \mathbf{c} \wr \wr)$, and $\mathtt{Next}_1(\wr \overline{s}, \mathbf{c} \wr \wr)$, for computing both branches of non-deterministic transitions. For deterministic transitions, we assume both branches are the same. Finally, we also assume we have a predicate $\mathtt{isPositive}(\wr \overline{s}, \mathbf{c} \wr \wr)$, which returns $\mathtt{true}$ if the configuration $\mathbf{c}$ is final and accepting, and $\mathtt{false}$ otherwise.

We define now, with substitutions, the following:

$$
\begin{aligned}
\mathtt{Accept}(\wr \varepsilon, \overline{s}, \mathbf{c} \wr \wr) &= \mathtt{isPositive}(\wr \overline{s}, \mathbf{c} \wr \wr) \\
\mathtt{Accept}(\wr \mathtt{s}(\overline{t}), \overline{s}, \mathbf{c} \wr \wr)) &= \mathtt{match\ \ IsUniversal}(\wr \overline{s}, \mathbf{c} \wr \wr)\ \mathtt{with} \\
&|\mathtt{true} \rightarrow \mathtt{AND}\ (\wr \mathtt{Accept}(\wr \overline{t}, \mathtt{Next}_0(\wr \overline{s}, \mathbf{c} \wr \wr), \mathbf{c} \wr \wr), \\
&\qquad \mathtt{Accept}(\wr \overline{t}, \mathtt{Next}_1(\wr \overline{s}, \mathbf{c} \wr \wr)) \wr \wr) \\
&|\mathtt{false} \rightarrow \mathtt{OR}\ (\wr \mathtt{Accept}(\wr \overline{t}, \mathtt{Next}_0(\wr \overline{s}, \mathbf{c} \wr \wr)), \mathbf{c} \wr \wr), \\
&\qquad \mathtt{Accept}(\wr \overline{t}, \mathtt{Next}_1(\wr \overline{s}, \mathbf{c} \wr \wr)) \wr \wr)\ .
\end{aligned}
$$

Then, for $\overline{t}$ and $\overline{s}$ large enough, and an initial configuration $\mathbf{c}$, $\mathtt{Accept}(\wr \overline{t}, \overline{s}, \mathbf{c} \wr \wr)$ outputs the result of the computation of the machine. Finally, nesting up to k layers of such recursion on values schemes allows, as in the proof of Theorem 7, to simulate a polynomial computation time.

The other direction is pretty straightforward: For any instance of a recursion scheme with substitutions, for any given values $\overline{r}$, $\mathbf{u}$, $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$, each bit of $\mathbf{g}(\mathbf{x} \wr \overline{r}, \mathbf{f}(\mathbf{x} \wr \overline{r}, k_1(\wr \mathbf{u} \wr \wr), \mathbf{y} \wr \mathbf{z} \wr \mathbf{t}), \mathbf{f}(\mathbf{x} \wr \overline{r}, k_2(\wr \mathbf{u} \wr \wr), \mathbf{y} \wr \mathbf{z} \wr \mathbf{t}), \mathbf{u}, \mathbf{y} \wr \mathbf{z} \wr \mathbf{t})$ is a boolean function of the bits of $\mathbf{f}(\mathbf{x} \wr \overline{r}, k_1(\wr \mathbf{u} \wr \wr), \mathbf{y} \wr \mathbf{z} \wr \mathbf{t})$ and $\mathbf{f}(\mathbf{x} \wr \overline{r}, k_2(\wr \mathbf{u} \wr \wr), \mathbf{y} \wr \mathbf{z} \wr \mathbf{t})$. Hence, it can be computed by an alternating procedure. The space bound follows from the bound on the size of the strings, provided by the tiering discipline. ◀

Recall now that the time bound in Theorem 9 follows from 7 by removing recursion on values from the algebra. The same applies here, as follows.

▶ **Definition 12** (NC-tropical functions). *The class of NC-tropical functions is the closure of our basic pointer tropical functions, under tropical composition, tropical recursion on notations and tropical recursion with substitutions on notations.*

▶ **Theorem 13.** *The class of NC-tropical functions with binary output is NC.*

Proof. The result follows from $A(logspace, polylogtime) = NC$ [18], and Theorem 7. Substitutions in the tropical recursion scheme on notations amounts to alternation. The proof is similar to that of Theorem 11, where additionally,

- The time bound on the computation of the machine needs only to be polylogarithmic, instead of polynomial. As in Theorem 9, tropical recursion on notations suffices to obtain this bound, and tropical recursion on values is no longer needed.
- For the other direction, any bit of $\mathbf{g}_a(\mathbf{x} \wr \overline{r}, \mathbf{f}(\mathbf{x} \wr \overline{r}, k_1(\wr \mathbf{u} \wr \wr), \mathbf{y} \wr \mathbf{z} \wr \mathbf{t}), \mathbf{f}(\mathbf{x} \wr \overline{r}, k_2(\wr \mathbf{u} \wr \wr), \mathbf{y} \wr \mathbf{z} \wr \mathbf{t}), \mathbf{u}, \mathbf{y} \wr \mathbf{z} \wr \mathbf{t})$ is again a boolean function of the bits of $\mathbf{f}(\mathbf{x} \wr \overline{r}, k_1(\wr \mathbf{u} \wr \wr), \mathbf{y} \wr \mathbf{z} \wr \mathbf{t})$ and $\mathbf{f}(\mathbf{x} \wr \overline{r}, k_2(\wr \mathbf{u} \wr \wr), \mathbf{y} \wr \mathbf{z} \wr \mathbf{t})$. Here, this boolean function can be computed by a boolean circuit of polylogarithmic depth, hence, by an alternating procedure in polylogarihtmic time. The arguments behind this remark are the same as the ones in the proof of $A(logspace, polylogtime) = NC$. ◀

7 Concluding Remarks

Theorems 7, 9, 11, and 13 rely on mutual recursive schemes. As stated above, we use these mutual schemes to express a space computation of size $k\log(n)$ for any constant k, with binary strings of length at most $\log(n) + c$. If we were to use only non-mutual recursion

schemes, we would need to have longer binary strings. This can be achieved by taking as input to our functions, not simply the `Offset`, but some larger string $\#^k(\texttt{Offset})$, where $\#^k$ is a function that appends k copies of its argument.

It also remains to be checked wether one can refine Theorem 13 to provide characterizations of the classes NC^i as in [14]. A first step in this direction is to define a recursion rank, accounting for the nesting of recursion schemes: then, check wether NC-tropical functions of rank i are computable in NC^i. Conversely, check also whether the simulation of Theorem 7 induces a fixed overhead, and wether NC^i can be encoded by NC-tropical functions of rank $i+c$ for some constant c small enough.

Finally, note that we characterize logarithmic space functions with logarithmically long output (Theorem 9), and NC functions with one-bit output (Theorem 13). As usual, polynomially long outputs for these classes can be retrieved via a pointer access: it suffices to parameterize these functions with an additional, logarithmically long input, denoting the output bit one wants to compute. In order to retrieve functions with polynomially long output, this approach could also be added to the syntax, with a `Write` construct similar to our `Read` construct, for writing the output.

References

1 Bill Allen. Arithmetizing Uniform NC. *Ann. Pure Appl. Logic*, 53(1):1–50, 1991. doi:10.1016/0168-0072(91)90057-S.

2 S. Bellantoni. *Predicative Recursion and Computational Complexity*. PhD thesis, University of Toronto, 1992.

3 Stephen Bellantoni and Stephen A. Cook. A New Recursion-Theoretic Characterization of the Polytime Functions. *Computational Complexity*, 2:97–110, 1992.

4 Stephen A. Bloch. Function-Algebraic Characterizations of Log and Polylog Parallel Time. *Computational Complexity*, 4:175–205, 1994. doi:10.1007/BF01202288.

5 Guillaume Bonfante, Reinhard Kahle, Jean-Yves Marion, and Isabel Oitavem. Two function algebras defining functions in NC^{k} boolean circuits. *Inf. Comput.*, 248:82–103, 2016. doi:10.1016/j.ic.2015.12.009.

6 Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981. doi:10.1145/322234.322243.

7 P. Clote. Sequential, machine-independent characterizations of the parallel complexity classes ALOGTIME, AC^k, NC^k and NC. *Feasible Mathematics, Birkhaüser, 49-69*, 1989.

8 A. Cobham. The intrinsic computational difficulty of functions. In Y. Bar-Hillel, editor, *Proceedings of the International Conference on Logic, Methodology, and Philosophy of Science*, pages 24–30. North-Holland, Amsterdam, 1962.

9 Martin Hofmann. Linear types and non-size-increasing polynomial time computation. *Inf. Comput.*, 183(1):57–85, 2003. doi:10.1016/S0890-5401(03)00009-9.

10 Martin Hofmann and Ulrich Schöpp. Pure pointer programs with iteration. *ACM Trans. Comput. Log.*, 11(4):26:1–26:23, 2010. doi:10.1145/1805950.1805956.

11 Satoru Kuroda. Recursion Schemata for Slowly Growing Depth Circuit Classes. *Computational Complexity*, 13(1-2):69–89, 2004. doi:10.1007/s00037-004-0184-4.

12 Daniel Leivant. A Foundational Delineation of Computational Feasiblity. In *Proceedings of the Sixth Annual Symposium on Logic in Computer Science (LICS '91), Amsterdam, The Netherlands, July 15-18, 1991*, pages 2–11. IEEE Computer Society, 1991. doi:10.1109/LICS.1991.151625.

13 Daniel Leivant and Jean-Yves Marion. Ramified Recurrence and Computational Complexity II: Substitution and Poly-Space. In Leszek Pacholski and Jerzy Tiuryn, editors, *CSL*, volume 933 of *Lecture Notes in Computer Science*, pages 486–500. Springer, 1994.

14 Daniel Leivant and Jean-Yves Marion. A characterization of alternating log time by ramified recurrence. *Theor. Comput. Sci.*, 236(1-2):193–208, 2000. doi:10.1016/S0304-3975(99)00209-1.

15 Daniel Leivant and Jean-Yves Marion. Ramified Recurrence and Computational Complexity IV : Predicative Functionals and Poly-Space. *Information and Computation*, page 12 p, 2000. to appear. Article dans revue scientifique avec comité de lecture. URL: https://hal.inria.fr/inria-00099077.

16 J. C. Lind. Computing in logarithmic space. Technical report, Massachusetts Institute of Technology, 1974.

17 Peter Møller Neergaard. A Functional Language for Logarithmic Space. In Wei-Ngan Chin, editor, *Programming Languages and Systems: Second Asian Symposium, APLAS 2004, Taipei, Taiwan, November 4-6, 2004. Proceedings*, volume 3302 of *Lecture Notes in Computer Science*, pages 311–326. Springer, 2004. doi:10.1007/978-3-540-30477-7_21.

18 Walter L. Ruzzo. On Uniform Circuit Complexity. *J. Comput. Syst. Sci.*, 22(3):365–383, 1981. doi:10.1016/0022-0000(81)90038-6.

Typed Equivalence of Effect Handlers and Delimited Control

Maciej Piróg
University of Wrocław, Poland
maciej.pirog@cs.uni.wroc.pl

Piotr Polesiuk
University of Wrocław, Poland
ppolesiuk@cs.uni.wroc.pl

Filip Sieczkowski
University of Wrocław, Poland
efes@cs.uni.wroc.pl

Abstract

It is folklore that effect handlers and delimited control operators are closely related: recently, this relationship has been proved in an untyped setting for deep handlers and the shift_0 delimited control operator. We positively resolve the conjecture that in an appropriately polymorphic type system this relationship can be extended to the level of types, by identifying the necessary forms of polymorphism, thus extending the definability result to the typed context. In the process, we identify a novel and potentially interesting type system feature for delimited control operators. Moreover, we extend these results to substantiate the folklore connection between shallow handlers and $\mathsf{control}_0$ flavour of delimited control, both in an untyped and typed settings.

2012 ACM Subject Classification Theory of computation → Control primitives; Theory of computation → Operational semantics; Software and its engineering → Polymorphism

Keywords and phrases type-and-effect systems, algebraic effects, delimited control, macro expressibility

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.30

Supplement Material The formalisation of results presented in this paper, performed in the Coq proof assistant, can be found at `https://pl-uwr.bitbucket.io/efftrans.zip`.

Funding *Maciej Piróg*: National Science Centre, Poland, POLONEZ 3 grant "Algebraic Effects and Continuations" no. 2016/23/P/ST6/02217.
This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No 665778.

Piotr Polesiuk: National Science Centre, Poland, grant no. 2014/15/B/ST6/00619.
Filip Sieczkowski: National Science Centre, Poland, grant no. 2016/23/D/ST6/01387.

Acknowledgements We would like to thank the anonymous reviewers of this paper for providing insightful comments and questions, particularly regarding presentation, as well as Dariusz Biernacki and Sam Lindley for helpful discussions of the technical content of the paper.

1 Introduction

Computational effects in programming languages are as pervasive as they are problematic. Virtually any programming language must allow its programmer some interaction with the outside world, at the very least through input and output channels, yet this concession tends to open Pandora's box of effects, built-in or user-defined, ranging from mutable state or logging mechanisms to complex nondeterminism or concurrency, and beyond. In most commonly used functional languages, such as OCaml or Scheme, control over occurrences

© Maciej Piróg, Piotr Polesiuk, and Filip Sieczkowski;
licensed under Creative Commons License CC-BY

4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019).
Editor: Herman Geuvers; Article No. 30; pp. 30:1–30:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of such effects is lax at best: this, however, leads to loss of strong reasoning principles, like β-equivalence, and invalidates many common program optimisations by making observational equivalence a very small relation.

Two main approaches to control and contain the rise of complexity brought by the presence of effects exist: monads [17, 21], as used in Haskell, is the more common in today's practice, but type-and-effect systems [15, 20] appear rather often in many of the more experimental programming languages. Both these approaches feature means of establishing that a program fragment is *pure*, which intuitively means that all the usual reasoning principles and program transformations should apply, and both allow the programmer to control which particular effects can arise in a given program fragment. Although the original type-and-effect systems were geared towards particular computational effects, such as mutable state, they have since been used also for languages with user-defined effects. In the following, we consider two families of such user-defined effectful abstraction: delimited control operators and algebraic effect handlers.

Delimited control operators were invented, in different flavours, by Felleisen [5] and Danvy and Filinski [4], and have been used to encode effects ever since (see for instance [3]). Algebraic effects and handlers, proposed by Plotkin and Pretnar [18] form an alternative approach to user-defined effects, generalising exceptions and their handlers in a more direct fashion than the delimited control operators, which arise more naturally by studying the shape of effectful programs in continuation-passing style. Both these families of abstractions admit diverse type systems: in this work we focus on a particular family of type-and-effect systems with effect rows, which allows a large degree of control over what effects the program performs and what is their order in the computation.

In this paper, we work within the program outlined by Forster *et al.* [8]: we study local syntax-directed translations (Felleisen's *macro translations* [6]) between *mutually expressible* extensions of a base calculus, thus studying the relationships between different modes of programming with user-defined effects. In this line, we make two contributions: first, we pinpoint the precise mode of polymorphism that is required for the macro translations between effect handlers and delimited control operators to preserve typings, and in the process we discover a novel extension of a type system for delimited control operators. Furthermore, we extend these results to the case of *shallow* effect handlers and the control_0 operator, thus substantiating the folklore claim that the shallow handlers behave in a "control-like" fashion. In contrast to Forster *et al.*, we focus solely on control operators with type-and-effect systems, leaving out the monadic approach to structuring effectful computations. The results presented in this paper are formalised using the Coq proof assistant.

2 The Common Core

In this paper, we discuss two calculi for delimited control operators (shift_0 and control_0) and two for effect handlers (deep and shallow), all of which are given as extensions of the core calculus defined in this section. This core calculus is a polymorphic λ-calculus with only basic infrastructure for effects that is shared by all four final calculi. For readability, we keep the calculi minimal, that is, they are equipped only with the features necessary to discuss control effects and their relationships. In some examples, however, we assume some basic types and constants, but this is only for readability, as they are clearly a feature orthogonal to control.

The syntax of the core calculus is given in Figure 1. The type-level variables are denoted $\alpha, \beta, \ldots$, while the term-level variables are denoted $f, r, x, y, \ldots$ The syntax of terms consists of variables, λ-abstractions, applications, and, as a part of the infrastructure for effects,

$$
\begin{aligned}
\mathrm{TVar} &\ni \alpha, \beta, \ldots && \text{(type-level variables)}\\
\mathrm{Var} &\ni f, r, x, y, \ldots && \text{(term-level variables)}\\
\mathrm{Kind} &\ni \kappa ::= \mathsf{T} \mid \mathsf{E} \mid \mathsf{R} && \text{(kinds)}\\
\mathrm{Typelike} &\ni \sigma, \tau, \varepsilon, \rho ::= \alpha \mid \tau \to_\rho \tau \mid \forall\, \alpha :: \kappa \,.\, \tau \mid \iota \mid \varepsilon \cdot \rho && \text{(types and rows)}\\
\mathrm{Val} &\ni u, v ::= x \mid \boldsymbol{\lambda} x\boldsymbol{.}\; e && \text{(values)}\\
\mathrm{Exp} &\ni e ::= v \mid e\; e \mid [e] && \text{(expressions)}\\
\mathrm{ECont} &\ni E ::= \square \mid E\; e \mid v\; E \mid [E] && \text{(evaluation contexts)}
\end{aligned}
$$

Figure 1 Syntax of the common core of the calculi.

$$
\frac{\alpha :: \kappa \in \Delta}{\Delta \vdash \alpha :: \kappa}
\qquad
\frac{\Delta \vdash \tau_1 :: \mathsf{T} \qquad \Delta \vdash \rho :: \mathsf{R} \qquad \Delta \vdash \tau_2 :: \mathsf{T}}{\Delta \vdash \tau_1 \to_\rho \tau_2 :: \mathsf{T}}
\qquad
\frac{\Delta, \alpha :: \kappa \vdash \tau :: \mathsf{T}}{\Delta \vdash \forall\, \alpha :: \kappa \,.\, \tau :: \mathsf{T}}
$$

$$
\frac{}{\Delta \vdash \iota :: \mathsf{R}}
\qquad
\frac{\Delta \vdash \varepsilon :: \mathsf{E} \qquad \Delta \vdash \rho :: \mathsf{R}}{\Delta \vdash \varepsilon \cdot \rho :: \mathsf{R}}
$$

Figure 2 Well-formedness of types and rows.

Biernacki *et al.*'s [2] *lift* operator $[e]$, which we discuss below. Note that the core calculus is given à la Curry, so the variable in a λ-abstraction is not labelled with a type. Moreover, we have the universal quantifier as a type former, but it is not reflected in the term-level syntax: generalisations and instantiations of the quantifier take place only in type derivations.

The core calculus is equipped with a type-and-effect system organised into three kinds: T for types, E for single effects, and R for rows of effects. Considering the well-formedness rules given in Figure 2, we read that the types are given by the mentioned universal quantifier (in which we quantify over variables of any kind), type variables, and arrows, which are decorated with rows of effects. Intuitively, a row of effects specifies what effects can happen when we evaluate an expression or call an effectful function (for a discussion on row-based type-and-effect systems see, for instance, [14]). Importantly, the specific calculi that we present in the subsequent sections do not distinguish effects by any sort of names, but by their position in the row associated to an expression. This means that the order of effects in a row is important and, intuitively, corresponds to the order of delimiters (handlers, resets) that can be placed around the expression – or, from another point of view, to the order in which a closing context introduces these effects. This is why we include the lift operator, as it allows us to manipulate the order of effects in a row and so, as discussed in Section 3.1, simulate a calculus with named effects. Formally, a row of effects can be given by a row variable, an empty row, written ι, or can be built by placing an effect in front of an existing row. In other words, each row is either *closed*, which means that it is a list of effects (in this case, we tend to omit the trailing ι when this does not lead to confusion), or *open*, which means that it is a list of effects that ends with a row variable. Note that the core calculus does not say much about single effects, except for the fact that we can quantify over effects as well as over types or rows. However, it captures how effects are organised into rows, as this aspect is common to all four final calculi.

$$\frac{}{\Delta \vdash \sigma <: \sigma} \qquad \frac{\Delta \vdash \tau_2^1 <: \tau_1^1 \qquad \Delta \vdash \rho_1 <: \rho_2 \qquad \Delta \vdash \tau_1^2 <: \tau_2^2}{\Delta \vdash \tau_1^1 \to_{\rho_1} \tau_1^2 <: \tau_2^1 \to_{\rho_2} \tau_2^2}$$

$$\frac{\Delta, \alpha :: \kappa \vdash \tau_1 <: \tau_2}{\Delta \vdash \forall \alpha :: \kappa \,.\, \tau_1 <: \forall \alpha :: \kappa \,.\, \tau_2} \qquad \frac{\Delta \vdash \rho :: \mathsf{R}}{\Delta \vdash \iota <: \rho} \qquad \frac{\Delta \vdash \rho_1 <: \rho_2}{\Delta \vdash \varepsilon \cdot \rho_1 <: \varepsilon \cdot \rho_2}$$

Figure 3 Subtyping.

$$\frac{x : \tau \in \Gamma}{\Delta; \Gamma \vdash x : \tau \;/\; \iota} \qquad \frac{\Delta \vdash \tau_1 :: \mathsf{T} \qquad \Delta; \Gamma, x : \tau_1 \vdash e : \tau_2 \;/\; \rho}{\Delta; \Gamma \vdash \boldsymbol{\lambda} x.\; e : \tau_1 \to_\rho \tau_2 \;/\; \iota}$$

$$\frac{\Delta; \Gamma \vdash e_1 : \tau_1 \to_\rho \tau_2 \;/\; \rho \qquad \Delta; \Gamma \vdash e_2 : \tau_1 \;/\; \rho}{\Delta; \Gamma \vdash e_1\; e_2 : \tau_2 \;/\; \rho} \qquad \frac{\Delta \vdash \varepsilon :: \mathsf{E} \qquad \Delta; \Gamma \vdash e : \tau \;/\; \rho}{\Delta; \Gamma \vdash [e] : \tau \;/\; \varepsilon \cdot \rho}$$

$$\frac{\Delta, \alpha :: \kappa; \Gamma \vdash e : \tau \;/\; \iota}{\Delta; \Gamma \vdash e : \forall \alpha :: \kappa \,.\, \tau \;/\; \iota} \qquad \frac{\Delta \vdash \sigma :: \kappa \qquad \Delta; \Gamma \vdash e : \forall \alpha :: \kappa \,.\, \tau \;/\; \rho}{\Delta; \Gamma \vdash e : \tau\{\sigma / \alpha\} \;/\; \rho}$$

$$\frac{\Delta \vdash \tau_1 <: \tau_2 \qquad \Delta \vdash \rho_1 <: \rho_2 \qquad \Delta; \Gamma \vdash e : \tau_1 \;/\; \rho_1}{\Delta; \Gamma \vdash e : \tau_2 \;/\; \rho_2}$$

Figure 4 Core typing rules.

Figure 3 shows the subtyping rules. The purpose of subtyping is to express the idea that one expression can perform “more” effects than another. In particular, it means that the empty row is a subtype of any other row, while two functions are in the relation if they satisfy the usual “contravariant-in-argument, covariant-in-result” condition together with the appropriate subtyping of their rows of effects.

With this, we can define the typing relation $\Delta; \Gamma \vdash e : \tau \;/\; \rho$, shown in Figure 4. It uses two contexts, Δ to assign a kind to a free type-level variable, and Γ to assign a type to a free term-level variable.[1] A term e is given a type τ and a row of effects ρ, which may be performed when evaluating e. The rules for variables, λ-abstractions, and applications are rather straightforward; the only deviation from the classic formulation of Talpin and Jouvelot [20] is that in the application rule we require the same effect on all premises, which is natural, as our system includes explicit subtyping. Additionally, we have the aforementioned rules that allow us to introduce and instantiate a universal quantifier, and a rule to apply subtyping. The rule for $[e]$ reveals some intuition about the purpose of the lift operator, as it allows us to add any effect in front of the row to “mask” the actual first effect. As hinted before, this is due to the fact that order of the effects matters: thus, we might be allowed to freely pretend that a given computation has more effects “outside” the ones that are actually present, via subeffecting, but we need to be explicit, and use an operator with some runtime content, when we want to manipulate effects at the *beginning* of the row. As it is much more natural to discuss the details of the semantics of lift with some actual effects at hand, we return to this discussion in more depth in Section 3.1, where we define deep effect handlers.

[1] We treat contexts as lists of bindings, allowing ourselves to write types of the form $\forall \Delta \,.\, \tau$, with the natural expansion as a chain of nested quantifiers of appropriate length.

$$\boldsymbol{\lambda} x.\, e\ v \mapsto e\{v\,/\,x\} \qquad [v] \mapsto v \qquad \frac{e_1 \mapsto e_2}{E[e_1] \to E[e_2]}$$

Figure 5 Single-step reduction and core contraction rules.

$$\frac{}{0\text{−free}(\square)} \qquad \frac{n\text{−free}(E)}{n\text{−free}(E\ e)} \qquad \frac{n\text{−free}(E)}{n\text{−free}(v\ E)} \qquad \frac{n\text{−free}(E)}{n+1\text{−free}([E])}$$

Figure 6 Freeness for core evaluation contexts.

Figure 5 together with the syntax of evaluation contexts from Figure 1 define the call-by-value operational semantics in terms of contractions, that is, single-step reductions in an evaluation context, in the standard style of Felleisen and Friedman [7]. Note that while the lift operator does not have any computational content on its own, it is used to manipulate the freeness of evaluation contexts defined in Figure 6. The freeness judgment was introduced by Biernacki et al. [2] to judgmentally match an effect handler to the operation, but it readily scales to the other systems, allowing us to move the lift to the common part of the calculus. Of course, since the core calculus does not contain any effect delimiters, the freeness can only increase, and only via the lift, but its use will become apparent when we consider particular effects.

3 Deep Effect Handlers and Delimited Control

We now turn to study the connection between the deep handlers and the shift_0 calculus of delimited control operators. The operational semantics of the calculi and their (untyped) equivalence are a slight variation on the ones studied by Forster et al. [8]. The correspondence between the type systems is our contribution.

3.1 Deep Handler Calculus

The calculus of deep effect handlers is given in Figure 7 as an extension of the core calculus. We extend the syntax of expressions with two forms: the first one, **do** v, is an operation with a single argument. Note that while the argument is necessarily a value, this restriction is purely a matter of convenience. The second new form is a handler, **handle** e $\{x, r.\ e;\ x.\ e\}$, in which the first expression is the handled computation, the $x, r.\ e$ part is an interpretation of the operation, while $x.\ e$ is the "return" clause, which describes the behaviour of the handler on effect-free computations. Handlers come with one new form of evaluation contexts, which allows reductions in the handled expression.

The reduction semantics for handlers is given by two contraction rules. The first one states that if we handle a 0-free context that has an operation in the evaluation position, the handler takes over, and proceeds according to the $x, r.\ e_h$ part, where e_h is the computation that we proceed with, in which x is bound to the value of the argument of the operation, while r is bound to the resumption, which allows us to continue evaluating the handled computation with a given value in place of the operation. Note that the resumption v_c is again wrapped in the same handler, which is why such handlers are called *deep*, as opposed

$$\sigma ::= \ldots \mid \Delta\,.\,\tau_1 \Rightarrow \tau_2$$

$$e ::= \ldots \mid \textbf{do}\ v \mid \textbf{handle}\ e\ \{x, r.\ e;\ x.\ e\}$$

$$E ::= \ldots \mid \textbf{handle}\ E\ \{x, r.\ e;\ x.\ e\}$$

$$\frac{\Delta, \Delta' \vdash \tau_1 :: \mathsf{T} \qquad \Delta, \Delta' \vdash \tau_2 :: \mathsf{T}}{\Delta \vdash \Delta'\,.\,\tau_1 \Rightarrow \tau_2 :: \mathsf{E}}$$

$$\frac{n+1\text{−free}(E)}{n\text{−free}(\textbf{handle}\ E\ \{x, r.\ e_h;\ x.\ e_r\})}$$

$$\frac{0\text{−free}(E) \qquad v_c = \boldsymbol{\lambda} z.\ \textbf{handle}\ E[z]\ \{x, r.\ e_h;\ x.\ e_r\}}{\textbf{handle}\ E[\textbf{do}\ v]\ \{x, r.\ e_h;\ x.\ e_r\} \mapsto e_h\{v \,/\, x\}\{v_c \,/\, r\}}$$

$$\textbf{handle}\ v\ \{x, r.\ e_h;\ x.\ e_r\} \mapsto e_r\{v \,/\, x\}$$

$$\frac{\Delta; \Gamma \vdash v : \delta(\tau_1) \,/\, \iota \qquad \Delta \vdash \delta :: \Delta' \qquad \Delta \vdash \Delta'\,.\,\tau_1 \Rightarrow \tau_2 :: \mathsf{E}}{\Delta; \Gamma \vdash \textbf{do}\ v : \delta(\tau_2) \,/\, (\Delta'\,.\,\tau_1 \Rightarrow \tau_2)}$$

$$\frac{\begin{array}{c}\Delta; \Gamma \vdash e : \tau \,/\, (\Delta'\,.\,\tau_1 \Rightarrow \tau_2) \cdot \rho \\ \Delta, \Delta'; \Gamma, x : \tau_1, r : \tau_2 \to_\rho \tau_r \vdash e_h : \tau_r \,/\, \rho \qquad \Delta; \Gamma, x : \tau \vdash e_r : \tau_r \,/\, \rho\end{array}}{\Delta; \Gamma \vdash \textbf{handle}\ e\ \{x, r.\ e_h;\ x.\ e_r\} : \tau_r \,/\, \rho}$$

Figure 7 Calculus of deep handlers.

to shallow handlers discussed in Section 4.1. The second single-step reduction rule is used when we handle a value. In such a case, the entire handler evaluates to e_r from the $x.\ e_r$ part, in which x is bound to the handled value.

A single effect in the type system is given as $\Delta\,.\,\tau_1 \Rightarrow \tau_2$, which, intuitively, specifies the type of the **do** operation. This means that its argument is of the type τ_1, while the operation applied to an argument can be used in contexts that expect an expression of the type τ_2. Importantly, each effect may bind any number of type variables of appropriate kinds, denoted by Δ, which means that the effect $\tau_1 \Rightarrow \tau_2$ can be polymorphic. One example of why such a feature is useful is the error effect, in which the operation is "raise an exception". Since we need to be able to do it regardless of the expected type, such an effect is captured by $\alpha :: \mathsf{T}\,.\,\mathit{unit} \Rightarrow \alpha$. For our purposes, such polymorphic effects are needed for the typed translations between deep handlers and the shift_0/reset delimited control operators. Such polymorphic operations have been considered before, for instance by Kammar et al. [11], and indeed such an extension to the type system with effects seems rather natural.

The typing rule for the **do** operation indeed matches the shape of the effect: the argument is required to have the type τ_1 and the entire expression has type τ_2. Moreover, we may pick any *type-level substitution* δ that is well-formed in Δ, and apply it to both types, thus *instantiating* the polymorphic operation. For example, in the error effect, if we want to raise an exception in a context in which we expect a value of the type *int*, we instantiate α to *int*. Formally, we define the well-formedness of a substitution as

$$\Delta \vdash \delta :: \Delta' \overset{\triangle}{=} \mathrm{dom}(\delta) = \mathrm{dom}(\Delta') \land \forall \alpha \in \mathrm{dom}(\delta), \Delta \vdash \delta(\alpha) :: \Delta'(\alpha).$$

The typing rule for $\textbf{handle}\ e\ \{x, r.\ e_h;\ x.\ e_r\}$ reveals that we handle the first effect in the row of the handled expression. The overall result of the handler has a type τ_r, which needs to be the type of both e_h and e_r. Importantly, the handler needs to be polymorphic in Δ', which means that for each particular occurrence of the operation, the expression e_h must be oblivious to the concrete instantiation of the variables of Δ'.

This type system is sound with respect to the operational semantics, as shown by the following result, obtained by the standard technique of progress and preservation (cf. Harper [9], Wright and Felleisen [22]:

▶ **Theorem 1.** *If $\cdot;\cdot \vdash e : \tau \,/\, \iota$ and $e \to^* e' \not\to$ in the calculus of deep handlers, then there exists a value v such that $e' = v$ and $\cdot;\cdot \vdash v : \tau \,/\, \iota$.*

One aspect that distinguishes our approach from most calculi for algebraic effects in the literature is that, for the simplicity of presentation, we have only one operation, **do**. Usually, one assumes a number of operations, often grouped in named effects, which is much more convenient in a programmer-level language, but is not necessary for our semantic considerations. This is the reason why we include the lift operator $[e]$ in our calculi, since it allows us to express programs with multiple effects. To see that this is the case, we now briefly discuss the relationship with multiple effects and multiple operations in an effect.

In most settings with multiple effects, each operation is associated with a particular effect [2, 13]. Then, operationally, it is assigned to the nearest enclosing handler of this effect. On the level of types, one then takes the rows $\varepsilon_1 \cdot \varepsilon_2 \cdot \rho$ and $\varepsilon_2 \cdot \varepsilon_1 \cdot \rho$ as equivalent for any two distinct effects ε_1 and ε_2, which allows freely swapping a pair of different effects in a row. Then, in a typing derivation, one needs to apply enough swaps to move the effect handled by a particular handler to the front of the row of the handled expression. However, as shown by Biernacki et al. [2], in a language with the lift operator, we can express a term-level swap function for any two positions in the row. For example, consider the expression $\mathbf{do}\ 2 + [\mathbf{do}\ ()] : \mathit{int} \,/\, (\,.\ \mathit{int}{\Rightarrow}\mathit{int}) \cdot (\,.\ \mathit{unit}{\Rightarrow}\mathit{int})$. Then, swapping the first two effects in the row would yield an expression equivalent to $[\mathbf{do}\ 2] + \mathbf{do}\ () : \mathit{int} \,/\, (\,.\ \mathit{unit}{\Rightarrow}\mathit{int}) \cdot (\,.\ \mathit{int}{\Rightarrow}\mathit{int})$. This means that our calculus can simulate a multiple-effect setting by manually placing enough lifts and swaps.

Another issue is the presence of multiple operations in an effect, which are handled together by a single handler. One can simulate this by tagging the argument of **do**, in a way that allows to distinguish between the operations. The difficulty is that different operations can have different types, depending on the tag. Thus, to simulate multiple operations in an effect, we could use GADTs, possibly encoded via equality types [23]. This aspect, however, is orthogonal to the aspects of control that we consider in this paper.

3.2 The shift_0 Delimited Control Calculus

The calculus of the shift_0 flavour of delimited control is given in Figure 8. It extends the core calculus with two new syntactic constructs: the $\mathbf{shift_0}$ operator and the reset delimiter, $\langle e | x.\ e_r\rangle$, where the expression e is the delimited computation, while $x.\ e_r$ is the "return" part.

Operationally, an expression $\mathbf{shift_0}\ k.\ e$ aborts the evaluation of the appropriate enclosing reset, replacing it with the expression e. However, the evaluation of the entire reset is not all-lost, as it is captured as a continuation bound to the variable k in e. If it happens that no $\mathbf{shift_0}$ is evaluated inside e in $\langle e | x.\ e_r\rangle$, the value of the entire reset is given by the value of e_r in which x is bound to the value of e.

The type-and-effect system is a generalisation of the calculus introduced by [8]. A single effect $\Delta\,.\ \tau \,/\, \rho$ can intuitively be seen as a specification of the type and effect of the continuation captured by any shift within the expression. Thus, if we consider the computation $\langle e | x.\ e_r\rangle$ and assume it has some type τ_r and effect row ρ_r, this type and row will be preserved as the first effect in the row associated with e. Then, in each $\mathbf{shift_0}\ k.\ e_s$ within e, the expression e_s is supposed to have this precise type and effect row – after all, the entire reset is replaced by e_s, so their types must agree. As a novel feature in our

$$\sigma ::= \ldots \mid \Delta\,.\,\tau\ /\ \rho$$

$$\frac{\Delta,\Delta' \vdash \tau :: \mathsf{T} \qquad \Delta,\Delta' \vdash \rho :: \mathsf{R}}{\Delta \vdash \Delta'\,.\,\tau\ /\ \rho :: \mathsf{E}}$$

$$e ::= \ldots \mid \textbf{shift}_0\, k.\ e \mid \langle e | x.\ e\rangle$$
$$E ::= \ldots \mid \langle E | x.\ e\rangle$$

$$\frac{n+1\text{−free}(E)}{n\text{−free}(\langle E|x.\ e\rangle)}$$

$$\frac{0\text{−free}(E) \qquad v_c = \boldsymbol{\lambda}\, z.\ \langle E[z]|x.\ e_r\rangle}{\langle E[\textbf{shift}_0\, k.\ e]|x.\ e_r\rangle \mapsto e\{v_c\,/\,k\}}$$

$$\langle v|x.\ e\rangle \mapsto e\{v\,/\,x\}$$

$$\frac{\begin{array}{c}\Delta,\Delta';\Gamma,k:\tau'{\to_\rho}\tau \vdash e : \tau\ /\ \rho' \\ \Delta,\Delta' \vdash \rho' <: \rho \qquad \Delta \vdash \tau' :: \mathsf{T} \qquad \Delta \vdash (\Delta'\,.\,\tau\ /\ \rho)\cdot\rho' :: \mathsf{R}\end{array}}{\Delta;\Gamma \vdash \textbf{shift}_0\, k.\ e : \tau'\ /\ (\Delta'\,.\,\tau\ /\ \rho)\cdot\rho'}$$

$$\frac{\Delta \vdash \delta :: \Delta' \qquad \Delta;\Gamma \vdash e : \tau'\ /\ (\Delta'\,.\,\tau\ /\ \rho)\cdot\delta(\rho) \qquad \Delta;\Gamma,x:\tau' \vdash e_r : \delta(\tau)\ /\ \delta(\rho)}{\Delta;\Gamma \vdash \langle e|x.\ e_r\rangle : \delta(\tau)\ /\ \delta(\rho)}$$

Figure 8 Calculus of shift_0/reset.

calculus, parts of the type and row captured as an effect by the reset may be abstracted: thus, both τ and ρ in an effect specification might depend on type variables bound by Δ – by the typing rule in Figure 8 we would have $\tau_r = \delta(\tau)$, and likewise for ρ. This can be intuitively understood as akin to existential quantification: the are concrete types known to a reset are abstracted within its body, but they are not revealed to the shifts within, which thus have to treat them parametrically, somewhat like an unpack operation.

This novel feature of the system should make us wonder if it is ever practical, particularly so given the long history of delimited control operators – and we believe it is. As an example, imagine a library that implements simple exceptions, with the signature given as two functions, $\mathsf{throw} : \forall\,\alpha :: \mathsf{T}\,.\ \mathit{unit}{\to_\varepsilon}\alpha$ and $\mathsf{try} : \forall\,\alpha :: \mathsf{T}\,.\ (\mathit{unit}{\to_\varepsilon}\alpha){\to}\alpha{\to}\alpha$,[2] for some, as yet unknown, effect ε. Can we express such a library using shift_0 and reset and, if so, what is the definition of ε? Clearly, throw needs to capture the context and discard it, and try should delimit the context – but how can throw procure the result it should return? In fact, it cannot produce such a result directly, since at definition site the type of the result is unknown – indeed, throw can (and should) be used in multiple contexts, where the return types expected by try are different.

This is where parametricity comes in. We can have throw capture the context and return an identity function, which will return whatever value the enclosing try would feed it, giving us $\mathsf{throw}\ () \triangleq \textbf{shift}_0\,_.\ \boldsymbol{\lambda}\, x.\ x$, which matches the signature given $\varepsilon \triangleq \alpha :: \mathsf{T}\,.\ \alpha{\to}\alpha\ /\ \iota$. Similarly, we have $\mathsf{try}\ \mathit{th}\ v \triangleq \langle \mathit{th}\ ()|x.\ \boldsymbol{\lambda}\,_.\ x\rangle\ v$, which also matches its signature with the given definition of ε. This simple example shows that this novel form of polymorphism gives us a certain separation of concerns, where the programmer may write effectful library functions and eliminate them using other constructs that insert reset delimiters in the appropriate places, instantiating the polymorphic effect at the same time.

[2] The second argument of try denotes the result of the entire expression if the exception was raised; this example could clearly be generalised, but as an illustration it is sufficient.

$$[\![\textbf{shift}_\mathbf{0}\, k.\ e]\!]^{\mathrm{DH}} \triangleq \textbf{do}\ (\boldsymbol{\lambda}\, k.\ [\![e]\!]^{\mathrm{DH}})$$
$$[\![\langle e | x.\ e_r\rangle]\!]^{\mathrm{DH}} \triangleq \textbf{handle}\ [\![e]\!]^{\mathrm{DH}}\ \{x, r.\ x\ r;\ x.\ [\![e_r]\!]^{\mathrm{DH}}\}$$
$$[\![\Delta\,.\ \tau\ /\ \rho]\!]^{\mathrm{DH}} \triangleq \alpha :: \mathsf{T}\,.\ (\forall\,\Delta\,.\ (\alpha \to_{[\![\rho]\!]^{\mathrm{DH}}} [\![\tau]\!]^{\mathrm{DH}}) \to_{[\![\rho]\!]^{\mathrm{DH}}} [\![\tau]\!]^{\mathrm{DH}}) \Rightarrow \alpha$$

Figure 9 Shift$_0$ as deep handlers.

Note that our calculus has additional constructs over the usual presentations of shift$_0$ and reset. In particular, we have the return clause in resets and the lift operator inherited from the core calculus. However, these constructs can be both macro-expressed in the standard setting for well-typed programs: the former was shown by Materzok and Biernacki [16], while the latter is definable as $[e] \triangleq \textbf{shift}_\mathbf{0}\, k.\ k\ e$.

Like with the deep handlers, we use progress and preservation lemmas to obtain the following soundness theorem:

▶ **Theorem 2.** *If $\cdot;\cdot \vdash e : \tau\ /\ \iota$ and $e \to^* e' \not\to$ in the calculus of shift$_0$ and reset, then there exists a value v such that $e' = v$ and $\cdot;\cdot \vdash v : \tau\ /\ \iota$.*

3.3 Typed Correspondence

We show the typed correspondence of the calculi presented in previous sections by presenting two local syntax-directed translations between calculi and showing that they preserve types and semantics. The translations of expressions are simple adaptations of those of Forster et al. [8], but the novel parts are translations of types and identifying which kind of polymorphism is required to obtain type preservation.

The translation from the calculus of shift$_0$ to deep handlers is shown in Figure 9. We only show translations of those parts of the language which need to be "macro expanded", i.e., the control constructs $\textbf{shift}_\mathbf{0}\, k.\ e$ and $\langle e | x.\ e_r\rangle$ at the level of expressions and the single effects at the level of types. For other constructs the translation behaves homomorphically: λ-abstraction is translated to a λ-abstraction, application to an application, etc.

The control operator **shift$_0$** performs en effect, so it is translated to **do** operation. The body of **shift$_0$** operator is expressed as a λ-abstraction and then passed as an argument to the operation, while a handler is responsible for providing the captured continuation. Indeed, the translation of reset follows this protocol: reset is turned into a handler which applies an argument of operation directly to the resumption.

In order to explain the translation of single effects, consider the following example. The expression $\langle E[\textbf{shift}_\mathbf{0}\, k.\ e] | x.\ e_r\rangle$ of type τ_0 where E is a 0-free evaluation context, will be translated into

$$\textbf{handle}\ ([\![E]\!]^{\mathrm{DH}}[\textbf{do}\ (\boldsymbol{\lambda}\, k.\ [\![e]\!]^{\mathrm{DH}})])\ \{x, r.\ x\ r;\ x.\ [\![e_r]\!]^{\mathrm{DH}}\},$$

so the effect handled by this handler should have the shape $((\tau' \to_\rho \tau) \to_\rho \tau) \Rightarrow \tau'$ where $\tau = [\![\tau_0]\!]^{\mathrm{DH}}$ and τ' is a translation of the type of $\textbf{shift}_\mathbf{0}\, k.\ e$ expression. The main difficulty of assigning types to this translation is that type τ' is not known by the handler, and can be different for any of the **shift$_0$** constructs, potentially even within the same reset: type τ' in a typing rule of **shift$_0$** does not occur in the effect. To solve this problem we use polymorphic effects and quantify over all possible τ' in the translated effect: thus, the result has the shape of $\alpha :: \mathsf{T}\,.\ ((\alpha \to_\rho \tau) \to_\rho \tau) \Rightarrow \alpha$.

$$[\![\textsf{do}\ v]\!]^{\mathrm{DD}} \triangleq \textsf{shift}_{\textsf{0}}\, k.\ \boldsymbol{\lambda} h.\ h\ [\![v]\!]^{\mathrm{DD}}\ (\boldsymbol{\lambda} x.\ k\ x\ h)$$

$$[\![\textsf{handle}\ e\ \{x, r.\ e_h;\ x.\ e_r\}]\!]^{\mathrm{DD}} \triangleq \langle [\![e]\!]^{\mathrm{DD}} | x.\ \boldsymbol{\lambda} h.\ [\![e_r]\!]^{\mathrm{DD}} \rangle\ (\boldsymbol{\lambda} x.\ \boldsymbol{\lambda} r.\ [\![e_h]\!]^{\mathrm{DD}})$$

$$[\![\Delta\,.\ \tau_1 \Rightarrow \tau_2]\!]^{\mathrm{DD}} \triangleq \alpha :: \mathsf{T}, \beta :: \mathsf{R}\,.\ \big((\forall\,\Delta\,.\ [\![\tau_1]\!]^{\mathrm{DD}} \to ([\![\tau_2]\!]^{\mathrm{DD}} \to_\beta \alpha) \to_\beta \alpha) \to_\beta \alpha\big)\ /\ \beta$$

Figure 10 Deep handlers as shift_0.

Polymorphic variables of the delimited-control effect (bound by Δ), behave, intuitively, as if they were existentially quantified, so they cannot be directly translated as polymorphic variables of the algebraic effect, which behave akin to universal quantification. Therefore, we express them as a chain of universal quantifiers on the left-hand-side of an arrow type: thus, the body of **shift₀** is polymorphic in Δ.

The translation preserves types and semantics, which is expressed by the following theorems.

▶ **Theorem 3.** *If $\Delta; \Gamma \vdash e : \tau\ /\ \rho$ in the calculus of shift_0, then $\Delta; [\![\Gamma]\!]^{\mathrm{DH}} \vdash [\![e]\!]^{\mathrm{DH}} : [\![\tau]\!]^{\mathrm{DH}}\ /\ [\![\rho]\!]^{\mathrm{DH}}$ in the calculus of deep handlers.*

▶ **Theorem 4** (after Forster et al.)**.** *If $e \to e'$ in the calculus of shift_0, then in the calculus of deep handlers we have $[\![e]\!]^{\mathrm{DH}} \to^+ [\![e']\!]^{\mathrm{DH}}$.*

The translation in the opposite direction is shown in Figure 10 and is slightly more complex. When the **do** v construct is reduced in the source language, the control is passed to the handler. After the translation, the **shift₀** operator needs to somehow obtain the code that "handles" the operation: thus, the body of **shift₀** immediately returns a λ-abstraction which expects the translated code of the handler as an argument. Therefore, the translation of the handler is itself more involved: it is translated into a reset *applied* to the body of the handler, itself expressed as a function.

Since the translation of a handler is not just a reset, but a reset applied to an argument, we have to take care about this argument in two places. First, the return clause of a handler is translated into a λ-abstraction which throws away the handler code. Secondly, the continuation k captured by the **shift₀** in the translation of **do** contains the reset, but does not contain the handler code. So the resumption passed to the h is not just k, but $\boldsymbol{\lambda} x.\ k\ x\ h$.

The type part of the translation of an effect $\Delta\,.\ \tau_1 \Rightarrow \tau_2$ is the type of a reset in the translation of a handler. This reset is a function which expects the translated handler clause of type $\forall\,\Delta\,.\ [\![\tau_1]\!]^{\mathrm{DD}} \to ([\![\tau_2]\!]^{\mathrm{DD}} \to_\beta \alpha) \to_\beta \alpha$ and returns α, where α is a type of the entire translated handler expression. The type α and the effect β of translated handler expression are not known at the site where **do** is performed, so we quantify over them in the translated effect, similarly to the previous translation. Now we can show that the translation preserves types.

▶ **Theorem 5.** *If $\Delta; \Gamma \vdash e : \tau\ /\ \rho$ holds in the calculus of deep handlers, then $\Delta; [\![\Gamma]\!]^{\mathrm{DD}} \vdash [\![e]\!]^{\mathrm{DD}} : [\![\tau]\!]^{\mathrm{DD}}\ /\ [\![\rho]\!]^{\mathrm{DD}}$ holds in the calculus of shift_0.*

As noted by Forster et al., we cannot obtain a direct analogue of Theorem 4. Instead, we let $\to_i$ be a relation on expressions in target calculus, defined by the following rule

$$\frac{e_1 \mapsto e_2}{C[e_1] \to_i C[e_2]},$$

where C is a general context with one hole, not necessarily in the evaluation position, and obtain the following theorem.

▶ **Theorem 6** (after Forster et al.). *If $e \to e'$ holds in the calculus of deep handlers, then $[\![e]\!]^{\mathrm{DD}} \to_i^+ [\![e']\!]^{\mathrm{DD}}$.*

4 Shallow Effect Handlers and Delimited Control

We now turn to the calculi of shallow handlers and $\mathrm{control}_0$. While these flavours of operators are arguably less popular than those studied in the previous section, they are nonetheless an interesting part of the spectrum of user-defined control operators. In the following, we formally capture the inter-expressibility of these two forms of delimited control, and comment on the differences with respect to the translations and results obtained in the case of deep handlers and shift_0.

4.1 Shallow Handler Calculus

Much like in the case of deep handlers, the calculus of shallow handlers is given in Figure 11. These handlers do not differ syntactically from their deep counterparts: the only difference lies within their semantics.

The operational semantics proceeds in a fashion that is very similar to the deep handlers: a **do** operation matches an enclosing handler using a freeness judgment, and both the value, and the captured continuation are passed to the handler. The only difference is that the captured continuation does *not* contain the handler itself – rather, only the evaluation context *within* the handler is captured. This lack of replication is often intuitively explained via the analogy to the case analysis and recursors, with the deep handlers' replication of the context making them behave in a more fold-like fashion [10].

Much like in the case of deep handlers, the calculus has a single effect constructor that denotes the type of the **do** operation. However, in addition to the polymorphic quantification that we have already seen in the previous section, this effect is *recursive*: it is prefixed with a form $\mu\alpha$, which binds α in the body of the effect as an effect-kinded variable. In both typing rules, when we access the "input" and "output" types of the effect, we substitute the entire effect for α in their body, thus effectively performing a single unfolding of the recursive effect. The only other difference with respect to the type system for the deep handlers lies in the type of the resumption in the rule for **handle**: since the resumption does not contain the handler, its return type is τ rather than τ_r, and it may still perform the same effect ε.

While recursive effect declarations abound in the literature, we are unaware of a prior formulation where the recursion is entirely confined to the type level, without making any appearance at the expression level. Note also that, much like polymorphic effects, the recursive effects are not necessary to establish soundness of the type system: we use them to establish inter-expressibility with the calculus of $\mathrm{control}_0$.

As with the previous calculi, we prove type soundness via progress and preservation:

▶ **Theorem 7.** *If $\cdot; \cdot \vdash e : \tau \mathrel{/} \iota$ and $e \to^* e' \not\to$ in the calculus of shallow handlers, then there exists a value v such that $e' = v$ and $\cdot; \cdot \vdash v : \tau \mathrel{/} \iota$.*

4.2 The $\mathrm{control}_0$ Delimited Control Calculus

Finally, we turn to the last of our calculi, the delimited control calculus of $\mathrm{control}_0$, presented in Figure 12. Much like the difference between deep and shallow handlers, the differences between various delimited control operators are rather subtle. The common intuition refers to

$$\sigma ::= \ldots \mid \mu\,\alpha\,.\,\Delta\,.\,\tau_1 \Rightarrow \tau_2$$

$$e ::= \ldots \mid \textbf{do}\ v \mid \textbf{handle}\ e\ \{x, r.\ e;\ x.\ e\}$$

$$E ::= \ldots \mid \textbf{handle}\ E\ \{x, r.\ e;\ x.\ e\}$$

$$\frac{\Delta, \alpha :: \mathsf{E}, \Delta' \vdash \tau_1 :: \mathsf{T} \qquad \Delta, \alpha :: \mathsf{E}, \Delta' \vdash \tau_2 :: \mathsf{T}}{\Delta \vdash \mu\,\alpha\,.\,\Delta'\,.\,\tau_1 \Rightarrow \tau_2 :: \mathsf{E}}$$

$$\frac{n+1\text{−free}(E)}{n\text{−free}(\textbf{handle}\ E\ \{x, r.\ e_h;\ x.\ e_r\})}$$

$$\frac{0\text{−free}(E)}{\textbf{handle}\ E[\textbf{do}\ v]\ \{x, r.\ e_h;\ x.\ e_r\} \mapsto e_h\{v / x\}\{\boldsymbol{\lambda} z.\ E[z] / r\}}$$

$$\textbf{handle}\ v\ \{x, r.\ e_h;\ x.\ e_r\} \mapsto e_r\{v / x\}$$

$$\frac{\varepsilon = \mu\,\alpha\,.\,\Delta'\,.\,\tau_1 \Rightarrow \tau_2 \qquad \delta' = \delta[\alpha \mapsto \varepsilon] \qquad \Delta; \Gamma \vdash v : \delta'(\tau_1)\ /\ \iota \qquad \Delta \vdash \delta :: \Delta' \qquad \Delta \vdash \varepsilon :: \mathsf{E}}{\Delta; \Gamma \vdash \textbf{do}\ v : \delta'(\tau_2)\ /\ \varepsilon}$$

$$\frac{\varepsilon = \mu\,\alpha\,.\,\Delta'\,.\,\tau_1 \Rightarrow \tau_2 \qquad \Delta; \Gamma \vdash e : \tau\ /\ \varepsilon \cdot \rho \qquad \Delta; \Gamma, x : \tau \vdash e_r : \tau_r\ /\ \rho \qquad \Delta, \Delta'; \Gamma, x : \tau_1\{\varepsilon / \alpha\}, r : \tau_2\{\varepsilon / \alpha\} \rightarrow_{\varepsilon \cdot \rho} \tau \vdash e_h : \tau_r\ /\ \rho}{\Delta; \Gamma \vdash \textbf{handle}\ e\ \{x, r.\ e_h;\ x.\ e_r\} : \tau_r\ /\ \rho}$$

Figure 11 Calculus of shallow handlers with recursive effects.

the treatment of the reset delimiters – see the account of Shan [19] for the complete picture, including shift and control. In our formulation, we keep the same shape of delimiters equipped with "return" clauses that we used with shift$_0$, and the control operator has an analogous form, while the operational semantics changes in the standard way, by not wrapping the captured continuation with delimiter.

The effect constructor for this calculus is the most complex of the ones we consider. In addition to the quantified variables (three, in this case: two types and a row) and the effect being recursive, like the one introduced for shallow effects, the underlying structure also contains an additional type. This is due to the mismatch between the return type of the continuation and the type of the expression under the control$_0$ operator: mirroring the case of shallow effects, this is caused by the fact that the captured continuation discards the return clause of the delimiter. Thus, leaving out the polymorphic quantifiers, in an effect $\tau_1 \Rightarrow \tau_2\ /\ \rho$ we have τ_1 as the type of the expression within the delimiter *and* the return type of the captured continuation, while τ_2 as the type of the return clause in the delimiter *and* the type of the expression under the control$_0$ operator.

As with the previous calculi, we prove type soundness via progress and preservation:

▶ **Theorem 8.** *If* $\cdot;\cdot \vdash e : \tau\ /\ \iota$ *and* $e \rightarrow^* e' \not\rightarrow$ *in the calculus of control*$_0$ *and reset, then there exists a value* v *such that* $e' = v$ *and* $\cdot;\cdot \vdash v : \tau\ /\ \iota$.

4.3 Typed Correspondence

The translation from the control$_0$ calculus to shallow handlers is presented in Figure 13. Note that the translation on expressions is precisely analogous to the case of shift$_0$ and deep handlers.

$$\sigma ::= \ldots \mid \mu\,\alpha\,.\,\Delta\,.\,\tau \Rightarrow \tau \;/\; \rho$$

$$e ::= \ldots \mid \mathbf{control_0}\, k.\; e \mid \langle e | x.\; e\rangle$$

$$E ::= \ldots \mid \langle E | x.\; e\rangle$$

$$\frac{\Delta, \alpha :: \mathsf{E}, \Delta' \vdash \tau_1 :: \mathsf{T} \qquad \Delta, \alpha :: \mathsf{E}, \Delta' \vdash \tau_2 :: \mathsf{T} \qquad \Delta, \alpha :: \mathsf{E}, \Delta' \vdash \rho :: \mathsf{R}}{\Delta \vdash \mu\,\alpha\,.\,\Delta'\,.\,\tau_1 \Rightarrow \tau_2 \;/\; \rho :: \mathsf{E}}$$

$$\frac{n+1\text{–free}(E)}{n\text{–free}(\langle E | x.\; e\rangle)}$$

$$\frac{0\text{–free}(E)}{\langle E[\mathbf{control_0}\, k.\; e] | x.\; e_r\rangle \mapsto e\{\boldsymbol{\lambda}\, z.\; E[z] \,/\, k\}} \qquad \langle v | x.\; e\rangle \mapsto e\{v \,/\, x\}$$

$$\frac{\varepsilon = \mu\,\alpha\,.\,\Delta'\,.\,\tau_1 \Rightarrow \tau_2 \;/\; \rho \qquad \Delta, \Delta'; \Gamma, k : \tau' \to_{\varepsilon\cdot\rho\{\varepsilon\,/\,\alpha\}} \tau_1\{\varepsilon \,/\, \alpha\} \vdash e : \tau_2\{\varepsilon \,/\, \alpha\} \;/\; \rho' \qquad \Delta, \Delta' \vdash \rho' <: \rho \qquad \Delta \vdash \tau' :: \mathsf{T} \qquad \Delta \vdash \varepsilon \cdot \rho' :: \mathsf{R}}{\Delta; \Gamma \vdash \mathbf{control_0}\, k.\; e : \tau' \;/\; \varepsilon \cdot \rho'}$$

$$\frac{\varepsilon = \mu\,\alpha\,.\,\Delta'\,.\,\tau_1 \Rightarrow \tau_2 \;/\; \rho \qquad \delta' = \delta[\alpha \mapsto \varepsilon] \qquad \tau_e = \delta'(\tau_1) \qquad \tau_r = \delta'(\tau_2) \qquad \rho_r = \delta'(\rho) \qquad \Delta \vdash \delta :: \Delta' \qquad \Delta; \Gamma \vdash e : \tau_e \;/\; \varepsilon \cdot \rho_r \qquad \Delta; \Gamma, x : \tau_e \vdash e_r : \tau_r \;/\; \rho_r}{\Delta; \Gamma \vdash \langle e | x.\; e_r\rangle : \tau_r \;/\; \rho_r}$$

Figure 12 Calculus of control$_0$ with recursive and polymorphic effects.

The new elements appear in the translation of the effect constructor. Like before, an effect $\tau_1 \Rightarrow \tau_2 \;/\; \rho$ after translation has the shape $\beta :: \mathsf{T}\,.\; ((\beta \to_{\rho_c} \tau_1') \to_{\rho'} \tau_2') \Rightarrow \beta$, where the ρ_c is a translated row of effects of the captured continuation. However, in this case the captured continuation does not contain the delimiter, so ρ_c is a nonempty row that contains the entire translated effect in its head position: this is why we need recursive effects to type-check the translated expressions. Now, the translated effect has the form $\mu\,\alpha\,.\,\beta :: \mathsf{T}\,.\,((\beta \to_{\alpha\cdot\rho'} \tau_1') \to_{\rho'} \tau_2') \Rightarrow \beta$. The translation of polymorphic variables of an effect is analogous to what we have seen in the case of "deep" control, with variables quantifiers of the effect turning into universal quantifiers in proper types.

We show the following theorems, which say that the translation preserves both types and semantics.

▶ **Theorem 9.** *If $\Delta; \Gamma \vdash e : \tau \;/\; \rho$ in the calculus of control$_0$, then $\Delta; [\![\Gamma]\!]^{\mathrm{SH}} \vdash [\![e]\!]^{\mathrm{SH}} : [\![\tau]\!]^{\mathrm{DH}} \;/\; [\![\rho]\!]^{\mathrm{SH}}$ in the calculus of shallow handlers.*

▶ **Theorem 10.** *If $e \to e'$ in the calculus of control$_0$, then in the calculus of shallow handlers we have $[\![e]\!]^{\mathrm{SH}} \to^{+} [\![e']\!]^{\mathrm{SH}}$.*

The translation from shallow handlers to the control$_0$ calculus is shown in Figure 14. The translation for expressions is almost the same as for the deep case with one minor difference: the continuation captured by $\mathbf{control_0}$ does not contain a delimiter, so it can be directly passed as a resumption to a handler code. The translation also requires recursive effects, for the same reasons as the translation in the other direction.

The translation preserves types and reduction semantics, as we show in the following theorems.

$$[\![\mathbf{control_0}\, k.\ e]\!]^{\mathrm{SH}} \triangleq \mathbf{do}\ \boldsymbol{\lambda} k.\ [\![e]\!]^{\mathrm{SH}}$$

$$[\![\langle e | x.\ e_r\rangle]\!]^{\mathrm{SH}} \triangleq \mathbf{handle}\ [\![e]\!]^{\mathrm{SH}}\ \{x, r.\ x\ r;\ x.\ [\![e_r]\!]^{\mathrm{SH}}\}$$

$$[\![\mu\, \alpha\, .\, \Delta\, .\, \tau_1 \Rightarrow \tau_2\ /\ \rho]\!]^{\mathrm{SH}} \triangleq \mu\, \alpha\, .\, \beta :: \mathsf{T}\, .\, \big(\forall\, \Delta\, .\, (\beta {\to_{\alpha\cdot[\![\rho]\!]^{\mathrm{SH}}}} [\![\tau_1]\!]^{\mathrm{SH}}) {\to_{[\![\rho]\!]^{\mathrm{SH}}}} [\![\tau_2]\!]^{\mathrm{SH}}\big) {\Rightarrow} \beta$$

Figure 13 The control$_0$ calculus via shallow handlers.

$$[\![\mathbf{do}\ v]\!]^{\mathrm{SD}} \triangleq \mathbf{control_0}\, k.\ \boldsymbol{\lambda} h.\ h\ [\![v]\!]^{\mathrm{SD}}\ k$$

$$[\![\mathbf{handle}\ e\ \{x, r.\ e_h;\ x.\ e_r\}]\!]^{\mathrm{SD}} \triangleq \langle [\![e]\!]^{\mathrm{SD}} | x.\ \boldsymbol{\lambda} h.\ [\![e_r]\!]^{\mathrm{SD}}\rangle\ (\boldsymbol{\lambda} x, r.\ [\![e_h]\!]^{\mathrm{SD}})$$

$$[\![\mu\, \alpha\, .\, \Delta\, .\, \tau_1 {\Rightarrow} \tau_2]\!]^{\mathrm{SD}} \triangleq$$
$$\mu\, \alpha\, .\, \beta_1 :: \mathsf{T}, \beta_2 :: \mathsf{T}, \gamma :: \mathsf{R}\, .\, \beta_1 \Rightarrow \big(\forall\, \Delta\, .\, [\![\tau_1]\!]^{\mathrm{SD}} {\to} ([\![\tau_2]\!]^{\mathrm{SD}} {\to_{\alpha\cdot\gamma}} \beta_1) {\to_\gamma} \beta_2\big) {\to_\gamma} \beta_2\ /\ \gamma$$

Figure 14 Shallow handlers as shallow delimited control.

▶ **Theorem 11.** *If $\Delta; \Gamma \vdash e : \tau\ /\ \rho$ in the calculus of shallow handlers, then $\Delta; [\![\Gamma]\!]^{\mathrm{SD}} \vdash [\![e]\!]^{\mathrm{SD}} : [\![\tau]\!]^{\mathrm{SD}}\ /\ [\![\rho]\!]^{\mathrm{SD}}$ in the calculus of control$_0$.*

▶ **Theorem 12.** *If $e \to e'$ in the calculus of shallow handlers, then in the calculus of control$_0$ we have $[\![e]\!]^{\mathrm{SD}} \to^+ [\![e']\!]^{\mathrm{SD}}$.*

Since we do not modify the captured continuation before passing it as a resumption, we are able to prove Theorem 12 for a reduction in evaluation contexts instead of general contexts.

5 Discussion and Further Work

We have shown how the untyped correspondence between deep effect handlers and shift$_0$, known from prior work can be extended to the typed setting, given an appropriately expressive type system. In the process, we have identified a novel type system for the shift$_0$/reset calculus, in which the shift expressions are parametric in the type and effect. To our knowledge, such a system has not been considered before in the extensive literature on delimited control operators, although further work is necessary to explore the relation of our system to those presently found in literature – one aspect to consider would certainly be answer-type modification.

At the same time, we believe it is useful to contrast the parametric shift$_0$, which remains rather difficult to grasp, to the apparently natural move to polymorphic effect handlers, which indeed have been considered before. We feel that the fact that the effect handlers give an interpretation to the control effect at the delimiter, rather than at the capture point – which causes the translations studied in this paper to have to “invert” the direction of control – may be the reason behind the recent surge in popularity of effect handlers as the main control abstraction provided by a language, while delimited control operators have been, throughout their history, somewhat of a niche interest.

We have also adapted the translations, for both terms and types, to the shallow handlers and a control_0/reset calculus. While the translation for terms is barely different from the case for deep handlers and shift_0, the type systems that are required in order to achieve inter-expressibility evidence the folklore notion that shallow handlers are "case-like" (while the deep variant behaves "fold-like"): thus the need for explicit recursion within the effects.

Finally, since we work within a common type-and-effect system with effect rows, we managed to extend the calculi with both a simple notion of subeffecting and a variant of Biernacki *et al.*'s lift operator that allows "masking" effects at the front of the effect row without any effect on the complexity of the translation.

While this work resolves a conjecture of Forster *et al.* [8], there remains a wide array of topics for future work. The most obvious line of work that we chose not to pursue at this point is answer-type modification [1, 12], which is a common feature of type systems for delimited control operators. Whether it can be reconciled with the polymorphic effects of our calculi remains to be investigated: if so, it is certainly interesting to see how it would translate to the effect handler calculi. Relating our type systems to the monadic representation is another aspect that would require further attention.

References

1. Kenichi Asai. On typing delimited continuations: three new solutions to the printf problem. *Higher-Order and Symbolic Computation*, 22(3):275–291, 2009. doi:10.1007/s10990-009-9049-5.
2. Dariusz Biernacki, Maciej Piróg, Piotr Polesiuk, and Filip Sieczkowski. Handle with care: relational interpretation of algebraic effects and handlers. *PACMPL*, 2(POPL):8:1–8:30, 2018. doi:10.1145/3158096.
3. Olivier Danvy. An analytical approach to programs as data objects. *DSc thesis, Department of Computer Science, Aarhus University*, 11, 2006.
4. Olivier Danvy and Andrzej Filinski. Abstracting Control. In *LISP and Functional Programming*, pages 151–160, 1990. doi:10.1145/91556.91622.
5. Matthias Felleisen. The Theory and Practice of First-Class Prompts. In *Conference Record of the Fifteenth Annual ACM Symposium on Principles of Programming Languages, San Diego, California, USA, January 10-13, 1988*, pages 180–190, 1988. doi:10.1145/73560.73576.
6. Matthias Felleisen. On the Expressive Power of Programming Languages. *Sci. Comput. Program.*, 17(1-3):35–75, 1991. doi:10.1016/0167-6423(91)90036-W.
7. Matthias Felleisen and Daniel P. Friedman. A Reduction Semantics for Imperative Higher-Order Languages. In *PARLE, Parallel Architectures and Languages Europe, Volume II: Parallel Languages, Eindhoven, The Netherlands, June 15-19, 1987, Proceedings*, pages 206–223, 1987. doi:10.1007/3-540-17945-3_12.
8. Yannick Forster, Ohad Kammar, Sam Lindley, and Matija Pretnar. On the expressive power of user-defined effects: effect handlers, monadic reflection, delimited control. *PACMPL*, 1(ICFP):13:1–13:29, 2017. doi:10.1145/3110257.
9. Robert Harper. *Practical Foundations for Programming Languages*. Cambridge University Press, New York, NY, USA, 2nd edition, 2016.
10. Daniel Hillerström and Sam Lindley. Shallow Effect Handlers. In *Programming Languages and Systems - 16th Asian Symposium, APLAS 2018, Wellington, New Zealand, December 2-6, 2018, Proceedings*, pages 415–435, 2018. doi:10.1007/978-3-030-02768-1_22.
11. Ohad Kammar, Sam Lindley, and Nicolas Oury. Handlers in action. In *ACM SIGPLAN International Conference on Functional Programming, ICFP'13, Boston, MA, USA - September 25 - 27, 2013*, pages 145–158, 2013. doi:10.1145/2500365.2500590.

12 Ikuo Kobori, Yukiyoshi Kameyama, and Oleg Kiselyov. Answer-Type Modification without Tears: Prompt-Passing Style Translation for Typed Delimited-Control Operators. In *Proceedings of the Workshop on Continuations, WoC 2016, London, UK, April 12th 2015.*, pages 36–52, 2015. doi:10.4204/EPTCS.212.3.

13 Daan Leijen. Type directed compilation of row-typed algebraic effects. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, pages 486–499, 2017. URL: http://dl.acm.org/citation.cfm?id=3009872.

14 Sam Lindley and James Cheney. Row-based effect types for database integration. In *Proceedings of TLDI 2012: The Seventh ACM SIGPLAN Workshop on Types in Languages Design and Implementation, Philadelphia, PA, USA, Saturday, January 28, 2012*, pages 91–102, 2012. doi:10.1145/2103786.2103798.

15 John M. Lucassen and David K. Gifford. Polymorphic Effect Systems. In *Conference Record of the Fifteenth Annual ACM Symposium on Principles of Programming Languages, San Diego, California, USA, January 10-13, 1988*, pages 47–57, 1988. doi:10.1145/73560.73564.

16 Marek Materzok and Dariusz Biernacki. A Dynamic Interpretation of the CPS Hierarchy. In *Programming Languages and Systems - 10th Asian Symposium, APLAS 2012, Kyoto, Japan, December 11-13, 2012. Proceedings*, pages 296–311, 2012. doi:10.1007/978-3-642-35182-2_21.

17 Eugenio Moggi. Computational Lambda-Calculus and Monads. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89), Pacific Grove, California, USA, June 5-8, 1989*, pages 14–23, 1989. doi:10.1109/LICS.1989.39155.

18 Gordon D. Plotkin and Matija Pretnar. Handling Algebraic Effects. *Logical Methods in Computer Science*, 9(4), 2013. doi:10.2168/LMCS-9(4:23)2013.

19 Chung-chieh Shan. A static simulation of dynamic delimited control. *Higher-Order and Symbolic Computation*, 20(4):371–401, 2007. doi:10.1007/s10990-007-9010-4.

20 Jean-Pierre Talpin and Pierre Jouvelot. The Type and Effect Discipline. *Inf. Comput.*, 111(2):245–296, 1994. doi:10.1006/inco.1994.1046.

21 Philip Wadler. Comprehending Monads. In *LISP and Functional Programming*, pages 61–78, 1990. doi:10.1145/91556.91592.

22 Andrew K. Wright and Matthias Felleisen. A Syntactic Approach to Type Soundness. *Inf. Comput.*, 115(1):38–94, 1994. doi:10.1006/inco.1994.1093.

23 Hongwei Xi, Chiyan Chen, and Gang Chen. Guarded recursive datatype constructors. In *Conference Record of POPL 2003: The 30th SIGPLAN-SIGACT Symposium on Principles of Programming Languages, New Orleans, Louisisana, USA, January 15-17, 2003*, pages 224–235, 2003. doi:10.1145/640128.604150.

Cubical Syntax for Reflection-Free Extensional Equality

Jonathan Sterling
Carnegie Mellon University, Pittsburgh, USA
http://cs.cmu.edu/~jmsterli
jmsterli@cs.cmu.edu

Carlo Angiuli
Carnegie Mellon University, Pittsburgh, USA
http://cs.cmu.edu/~cangiuli
cangiuli@cs.cmu.edu

Daniel Gratzer
Aarhus University, Denmark
http://jozefg.github.io
gratzer@cs.au.dk

Abstract

We contribute XTT, a cubical reconstruction of Observational Type Theory [7] which extends Martin-Löf's intensional type theory with a *dependent equality type* that enjoys function extensionality and a judgmental version of the *unicity of identity proofs* principle (UIP): any two elements of the same equality type are judgmentally equal. Moreover, we conjecture that the typing relation can be decided in a practical way. In this paper, we establish an algebraic canonicity theorem using a novel extension of the *logical families* or *categorical gluing* argument inspired by Coquand and Shulman [27, 48]: every closed element of boolean type is derivably equal to either true or false.

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

Keywords and phrases Dependent type theory, extensional equality, cubical type theory, categorical gluing, canonicity

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.31

Related Version An extended version is available at https://arxiv.org/abs/1904.08562.

Funding The authors gratefully acknowledge the support of the Air Force Office of Scientific Research through MURI grant FA9550-15-1-0053. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the AFOSR.

Acknowledgements We thank Lars Birkedal, Evan Cavallo, David Thrane Christiansen, Thierry Coquand, Kuen-Bang Hou (Favonia), Marcelo Fiore, Jonas Frey, Krzysztof Kapulkin, András Kovács, Dan Licata, Conor McBride, Darin Morrison, Anders Mörtberg, Michael Shulman, Bas Spitters, and Thomas Streicher for helpful conversations about extensional equality, algebraic type theory, and categorical gluing. We thank our anonymous reviewers for their insightful comments, and especially thank Robert Harper for valuable conversations throughout the development of this work. We also thank Paul Taylor for his diagrams package, which we have used to typeset the commutative diagrams in this paper.

© Jonathan Sterling, Carlo Angiuli, and Daniel Gratzer;
licensed under Creative Commons License CC-BY

1 Introduction

The past fifty years of constructive type theory can be summed up as the search for a scientific understanding of *equality*, punctuated by moments of qualitative change in our perception of the boundary between semantics (actual construction) and syntax (proof theory) from a type-theoretic point of view. Computation is critical to both the semantics and syntax of type theory – from Martin-Löf's meaning explanations [43], supplying type theory with its direct semantics and intuitionistic grounding, to syntactic properties such as closed and open *canonicity* which establish computation as the indispensible method for deriving equations.

For too long, a limiting perspective on extensional type theory has prevailed, casting it as a particular syntactic artifact (for instance, the formalism obtained by stripping of their meaning the rules which incidentally appear in Martin-Löf's monograph [43]), a formal system which enjoys precious few desirable syntactic properties and is distinguished primarily by its *equality reflection* rule.

We insist on the contrary that the importance of extensional type theory lies not in the specific choice of syntactic presentation (historically, via equality reflection), but rather in the *semantic* characteristics of its equality connective, which are invariant under choice of syntax. The specifics of how such an equality construct is presented syntactically are entirely negotiable (the internal language of a doctrine is determined only up to equivalence), and therefore has an empirical component.

1.1 Internalizing equality: from judgments to types

Equality in type theory begins with a form of judgment $\Gamma \vdash A = B$ *type*, which expresses that A and B are exactly the same type; because types can depend on terms, one also includes a form of judgment $\Gamma \vdash M = N : A$ to express that M and N are exactly the same element of A. This kind of equality, called *judgmental equality*, is silent in the sense that if $\Gamma \vdash A = B$ *type* holds and $\Gamma \vdash M : A$ holds, then $\Gamma \vdash M : B$ without further ado.

Judgmental equality in type theory is a completely top-level affair: it cannot be assumed or negated. On the other hand, both programming and mathematics require one to establish equations under the assumption of other equations (for instance, as part of an induction hypothesis). For this reason, it is necessary to *internalize* the judgmental equality $M = N : A$ as a type $\mathsf{Eq}_A(M, N)$ which can be assumed, negated, or inhabited by induction.

The simplest way to internalize judgmental equality as a type is to provide introduction and elimination rules which make the existence of a proof of $\mathsf{Eq}_A(M, N)$ equivalent to the judgment $M = N : A$:

INTRODUCTION
$$\frac{\Gamma \vdash M = N : A}{\Gamma \vdash \mathsf{refl} : \mathsf{Eq}_A(M, N)}$$

ELIMINATION
$$\frac{\Gamma \vdash P : \mathsf{Eq}_A(M, N)}{\Gamma \vdash M = N : A}$$

The ELIMINATION rule above is usually called *equality reflection*, and is characteristic of *extensional* versions of Martin-Löf's type theory. This presentation of the equality type is very strong, and broadens the reach of judgmental equality into assertions of higher-level.

A consequence of the equality reflection rule is that judgmental equality is no longer decidable, a pragmatic concern which affects implementation and usability. On the other hand, equality reflection implies numerous critical reasoning principles, including function extensionality (if two functions agree on all inputs, then they are equal), a judgmental version of the famous *unicity of identity proofs* (UIP) principle (any two elements of the equality type are equal), and perhaps the most crucial consequence of internalized equality, *coercion* (if $M : P(a_0)$ and $P : \mathsf{Eq}_A(a_0, a_1)$, then there is some term $P^*(M) : P(a_1)$; in this case, $P^*(M) = M$).

1.2 Extensional equality via equality reflection

The earliest type-theoretic proof assistants employed the equality reflection rule (or equivalent formulations) in order to internalize the judgmental equality, a method most famously represented by Nuprl [25] and its descendents, including RedPRL [10]. The Nuprl-style formalisms act as a "window on the truth" for a *single* intended semantics inspired by Martin-Löf's computational meaning explanations [2]; semantic justification in the computational ontology is the *only* consideration when extending the Nuprl formalism with a new rule, in contrast to other traditions in which global properties (e.g. admissibility of structural rules, decidability of typing, interpretability in multiple models, etc.) are treated as definitive.

Rather than supporting *type checking*, proof assistants in this style rely heavily on interactive development of typing derivations using tactics and partial decision procedures. A notable aspect of the Nuprl family is that their formal sequents range not over typed terms (proofs), but over untyped raw terms (realizers); a consequence is that during the proof process, one must repeatedly establish numerous *type functionality* subgoals, which restore the information that is lost when passing from a proof to a realizer. To mitigate the corresponding blow-up in proof size, Nuprl relies heavily on untyped computational reasoning via pointwise functionality, a non-standard semantics for dependently typed sequents which has some surprising consequences, such as refuting the principle of *dependent cut* [38].

Another approach to implementing type theory with equality reflection is exemplified in the experimental Andromeda proof assistant [15], in which proofs are also built interactively using tactics, but judgments range over abstract proof derivations rather than realizers. This approach mitigates to some degree the practical problems caused by erasing information prematurely, and also enables interpretation into a broad class of semantic models.

Although Nuprl/RedPRL and Andromeda illustrate that techniques beyond mere type checking are profitable to explore, the authors' experiences building and using RedPRL for concrete formalization of mathematics underscored the benefits of having a practical algorithm to check types, particularly in the setting of cubical type theory (Section 1.6), whose higher-dimensional structure significantly reduces the applicability of Nuprl-style untyped reasoning.

In particular, whereas it is possible to treat *all* β-rules and many η-rules in non-cubical type theory as untyped rewrites, such an approach is unsound for the cubical account of higher inductive types and univalence [11]; consequently, in RedPRL many β/η rewrites must emit auxiliary proof obligations. Synthesizing these experiences and challenges led to the creation of the `redtt` proof assistant for Cartesian cubical type theory [9].

1.3 Equality in intensional type theory

Martin-Löf's Intensional Type Theory (ITT) [41, 46] represents another extremal point in the internalization of judgmental equality. ITT underapproximates the equality judgment via its *identity type*, characterized by rules like the following:

FORMATION
$$\frac{\Gamma \vdash A\ \mathit{type} \qquad \Gamma \vdash M : A \qquad \Gamma \vdash N : A}{\Gamma \vdash \mathsf{Id}_A(M, N)\ \mathit{type}}$$

INTRODUCTION
$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \mathsf{refl}_A(M) : \mathsf{Id}_A(M, M)}$$

ELIMINATION
$$\frac{\Gamma, x : A, y : A, z : \mathsf{Id}_A(x, y) \vdash C(x, y, z)\ \mathsf{type} \qquad \Gamma \vdash P : \mathsf{Id}_A(M, N) \qquad \Gamma, x : A \vdash Q : C(x, x, \mathsf{refl}_A(x))}{\Gamma \vdash \mathsf{J}_{x,y,z.C}(P; x.Q) : C(M, N, P)}$$

$\ldots$

Symmetry, transitivity and coercion follow from the elimination rule of the identity type. Other properties which follow directly from equality reflection, such as the unicity of identity proofs and function extensionality, are not validated by ITT; indeed, there are sufficiently intensional models of the identity type to refute both properties [52, 33]. While the desirability of the unicity principle is perhaps up for debate, especially in light of recent developments in Homotopy Type Theory [55], theorists and practitioners alike generally agree that function extensionality is desirable.

A significant selling-point for ITT is that, by avoiding equality reflection, it presents a theory which can be implemented using type checking and normalization. Consequently, β and η rules are totally automatic and never require intervention from the user – in contrast to systems like RedPRL, whose users are accustomed to establishing β/η equivalences by hand at times when heuristical tactics prove inadequate. The downsides of pure ITT, however, are manifold: function extensionality is absolutely critical in practice.

1.4 Setoids and internal model constructions

A standard technique for avoiding the deficiencies of the identity type in ITT is the *setoid construction* [32], an exact completion which glues an equivalence relation $=_A$ onto each type $|A|$ in the spirit of Bishop [16]. When using setoids, a function $A \to B$ consists of a type-theoretic function $f : |A| \to |B|$ together with a proof that it preserves the equivalence relation, $f_= : (x, y : |A|) \to x =_A y \to f(x) =_B f(y)$; a *dependent setoid* (family of setoids) is a type-theoretic family equipped with a coherent coercion operator.

Setoids are a discipline for expressing internally precisely the extrinsic properties required for constructions to be extensional (compatible with equality); these extra proof obligations must be satisfied in parallel with constructions at every turn. The state of affairs for setoids is essentially analogous to that of proof assistants with equality reflection, in which type functionality subgoals play a similar role to the auxiliary paperwork generated by setoids.

Paradoxically, however, every construction in ordinary ITT is automatically extensional in this sense. A solution to the problem of equality in type theory should, unlike setoids, take advantage of the fact that type theory is already restricted to extensional constructions, adding to it only enough language to refer to equality internally. This is the approach taken by both Observational Type Theory and XTT.

1.5 Observational Type Theory

The first systematic solution to the problem of syntax for extensional equality without equality reflection was Observational Type Theory (OTT) [6, 7], which built on early work by Altenkirch and McBride [3, 44]. The central idea of OTT is to work with a *closed* universe of types, defining by recursion for each pair of types A, B a type $\mathsf{Eq}(A, B)$ of proofs that A and B are equal, and for each pair of elements $M : A$ and $N : B$, a type of proofs $\mathsf{Eq}_{A,B}(M, N)$ that M and N are (heterogeneously) equal. Finally, one defines "generic programs" by recursion on type structure which calculate coercions and coherences along proofs of equality.

One can think of OTT as equipping the semantic setoid construction with a direct-style type-theoretic language, and adding to it closed, inductively defined universes of types. The heterogeneous equality of OTT, initially a simplifying measure adopted from McBride's thesis [44], is an early precursor of the *dependent paths* which appear in Homotopy Type Theory [55], Cubical Type Theory [24, 8, 11], and XTT.

Recently, McBride and his collaborators have made progress toward a cubical version of OTT, using a different cube category and coercion structure, in which one coerces only from 0 to 1, and obtains fillers using an affine rescaling operation [23].

1.6 Cubical Type Theory

In a rather different line of research, Voevodsky showed that Intensional Type Theory is compatible with a *univalence axiom* yielding an element of $\mathsf{Id}_{\mathcal{U}}(A, B)$ for every equivalence (coherent isomorphism) between types A, B [37, 55]. A univalent universe *classifies* types under a certain size cut-off in the sense of higher topos theory [40]. However, Intensional Type Theory extended with univalence lacks *canonicity*, because identity elimination computes only on refl and not on proofs constructed by univalence.

Since then, *cubical type theories* have been developed to validate univalence without disrupting canonicity [24, 11]. These type theories extend Martin-Löf's type theory with an abstract interval, maps out of which represent paths, a higher-dimensional analogue to equality; the interval has abstract elements, represented by a new sort of *dimension* variable i, and constant endpoints $0, 1$. Coercions arise as an instance of *Kan structure* governed directly by the structure of paths between types, which are nothing more than types dependent on an additional dimension variable.

There are currently two major formulations of cubical type theory. De Morgan cubical type theory [24] equips the interval with negation and binary connection (minimum and maximum) operations. Cartesian cubical type theory [8, 11], the closest relative of XTT, has no additional structure on the interval, but equips types with a much stronger notion of coercion generalizing the one described in Section 2.1.1.

1.7 Our contribution: XTT

We contribute XTT (Appendix A), a new type theory that supports extensional equality without equality reflection, using ideas from cubical type theory [24, 8, 11]. In particular, we obtain a compositional account of propositional equality satisfying function extensionality and a *judgmental* version of the unicity of identity proofs – when $P, Q : \mathsf{Eq}_A(M, N)$, we have $P = Q$ judgmentally – enabling us to substantially simplify our Kan operations (Section 2.1.2). Moreover, XTT is closed under a cumulative hierarchy[1] of closed universes à la Russell. We hope to integrate XTT into the redtt cubical proof assistant [9] as an implementation of extensional equality in the style of two-level type theory [11].

A common thread that runs through the XTT formalism is the decomposition of constructs from OTT into more modular, *judgmental* principles. For instance, rather than defining equality separately at every type and entangling the connectives, we define equality once and for all using the interval. Likewise, rather than ensuring that equality proofs are unique through brute force, we obtain unicity using a structural rule which does not mention the equality type.

By first developing the model theory of XTT in an algebraic way (Section 3), we then prove a canonicity theorem for the *initial* model of XTT (Section 3.2): any closed term of boolean type is equal to either true or false. This result is obtained using a novel extension of the categorical gluing technique described by Coquand and Shulman [27, 48]. Canonicity expresses a form of "computational adequacy" – in essence, that the equational theory of XTT suffices to derive any equation which ought to hold by (closed) computation – and is one of many syntactical considerations that experience has shown to be correlated to usability.

[1] As in previous work [49], we employ an *algebraic* version of cumulativity which does not require subtyping.

(cubes)	Ψ, Φ	$::=$	$\cdot \mid \Psi, i \mid \Psi, \xi$
(contexts)	Γ, Δ	$::=$	$\cdot \mid \Gamma, x : A$
(dimensions)	r, s	$::=$	$i \mid \varepsilon$
(constant dims.)	ε	$::=$	$0 \mid 1$
(constraints)	ξ	$::=$	$r = r'$
(universe levels)	k, l	$::=$	$n \quad (n \in \mathbb{N})$
(types)	A, B	$::=$	$M \mid (x : A) \to B \mid (x : A) \times B \mid \mathsf{Eq}_{i.A}(M, N) \mid \Uparrow^{l}_{k} A \mid \mathcal{U}_k \mid \mathsf{bool}$
(terms)	M, N	$::=$	$x \mid A \mid \lambda x.M \mid \mathsf{app}_{x:A.B}(M, N) \mid \langle M, N \rangle \mid \mathsf{fst}_{x:A.B}(M) \mid \mathsf{snd}_{x:A.B}(M) \mid$ $\lambda i.M \mid \mathsf{app}_{i.A}(M, r) \mid \mathsf{true} \mid \mathsf{false} \mid \mathsf{if}_{x.A}(M; N_0, N_1) \mid$ $[i.A] \downarrow^{r}_{r'} M \mid A \downarrow^{r}_{r'} M \; [s \text{ with } 0 \hookrightarrow j.N_0 \mid 1 \hookrightarrow j.N_1]$

Figure 1 A summary of the raw syntax of XTT. As a matter of top-level notation, we freely omit annotations that can be inferred from context, writing $M(N)$ for $\mathsf{app}_{x:A.B}(M, N)$. The annotations chosen in the raw syntax are the minimal ones required to establish a coherent interpretation into the initial XTT-algebra; for instance, it is unnecessary to include an annotation on the λ-abstraction.

2 Programming and proving in XTT

Like other cubical type theories, the XTT language extends Martin-Löf's type theory with a new sort of variable i ranging over an abstract interval with global elements 0 and 1; we call an element r of the interval a *dimension*, and we write ε to range over a constant dimension 0 or 1. Cubical type theories like XTT also use a special kind of hypothesis to constrain the values of dimensions: when r and s are dimensions, then $r = s$ is a *constraint*. In XTT, a single context Ψ accounts for both dimension variables (Ψ, i) and constraints $(\Psi, r = s)$. We will write $\Psi \mid r$ *dim* for when a dimension r is valid in a dimension context Ψ. The judgment $\Psi \mid r = s$ *dim* holds when r and s are equal as dimensions with respect to the constraints in Ψ. Dimensions can be substituted for dimension variables, an operation written $M\langle r/i \rangle$.

Finally, ordinary type-theoretic assumptions $x : A$ are kept in a context Γ that depends on Ψ. In XTT, a full context is therefore written $\Psi \mid \Gamma$. The meaning of a judgment at context $(\Psi, i = r)$ is completely determined by its instance under the substitution r/i. Under the false constraint $0 = 1$, all judgments hold; the resulting collapse of the typing judgment and the judgmental equality does not disrupt any important metatheoretic properties, because the theory of dimensions is decidable.

The general typehood judgment $\Psi \mid \Gamma \vdash A \; \mathit{type}_k$ means that A is a type of universe level k in context Γ over the cube Ψ; note that this judgment presupposes the well-formedness of Ψ, Γ. Likewise, the element typing judgment $\Psi \mid \Gamma \vdash M : A$ means that M is an element of the type A in Γ over Ψ as above; this form of judgment presupposes the well-formedness of A and thence Ψ, Γ. We also have typed judgmental equality $\Psi \mid \Gamma \vdash A = B \; \mathit{type}_k$ and $\Psi \mid \Gamma \vdash M = N : A$, which presuppose the well-formedness of *all* their constituents.

Dependent equality types

XTT extends Martin-Löf type theory with *dependent equality types* $\mathsf{Eq}_{i.A}(N_0, N_1)$ when $\Psi, i \mid \Gamma \vdash A \; \mathit{type}_k$ and $\Psi \mid \Gamma \vdash N_0 : A\langle 0/i \rangle$ and $\Psi \mid \Gamma \vdash N_1 : A\langle 1/i \rangle$. Geometrically, elements of this type are *lines* or *paths* in the type A ranging over dimension i, with left endpoint N_0 and right endpoint N_1.[2] This type captures internally the equality of N_0 and N_1; dependency

[2] Our dependent equality types are locally the same as dependent path types $\mathsf{Path}_{i.A}(N_0, N_1)$ from cubical type theories; however, we have arranged in XTT for them to satisfy a unicity principle by which

of A on the dimension i is in essence a cubical reconstruction of heterogeneous equality, albeit with different properties from the version invented by McBride in his thesis [44].

An element of the equality type $\mathsf{Eq}_{i.A}(N_0, N_1)$ is formed by the dimension λ-abstraction $\lambda i.M$, requiring that M is an element of A in the extended context, and that N_0, N_1 are the left and right sides of M respectively. Proofs P of equality are eliminated by dimension application, $P(r)$, and are subject to β, η, ξ rules analogous to those for function types. Finally, we have $P(\varepsilon) = N_\varepsilon$ always, extending Gentzen's principle of inversion to the side condition that we placed on M. More formally:

$$\frac{\Psi, i \mid \Gamma \vdash M : A \qquad \overrightarrow{\Psi, i = \varepsilon \mid \Gamma \vdash M = N_\varepsilon : A}}{\Psi \mid \Gamma \vdash \lambda i.M : \mathsf{Eq}_{i.A}(N_0, N_1)} \qquad \frac{\Psi \mid r\ dim \qquad \Psi \mid \Gamma \vdash M : \mathsf{Eq}_{i.A}(N_0, N_1)}{\Psi \mid \Gamma \vdash M(r) : A\langle r/i\rangle} \qquad \frac{\Psi \mid \Gamma \vdash M : \mathsf{Eq}_{i.A}(N_0, N_1)}{\Psi \mid \Gamma \vdash M(\varepsilon) = N_\varepsilon : A\langle \varepsilon/i\rangle}$$

$$\frac{\Psi \mid \Gamma \vdash M : \mathsf{Eq}_{i.A}(N_0, N_1)}{\Psi \mid \Gamma \vdash M = \lambda i.M(i) : \mathsf{Eq}_{i.A}(N_0, N_1)} \qquad \frac{\Psi, i \mid \Gamma \vdash M : A}{\Psi \mid \Gamma \vdash (\lambda i.M)(r) = M\langle r/i\rangle : A\langle r/i\rangle}$$

Function extensionality

A benefit of the cubical formulation of equality types is that the principle of function extensionality is trivially derivable in a computationally well-behaved way. Suppose that $f, g : (x : A) \to B$ and we have a family of equalities $h : (x : A) \to \mathsf{Eq}_{_.B}(f(x), g(x))$; then, we obtain a proof that f equals g by abstraction and application:

$$\lambda i.\lambda x.h(x)(i) : \mathsf{Eq}_{_.(x:A)\to B}(f, g)$$

In semantics of type theory, the structure of equality on a type usually mirrors the structure of the *elements* of that type in a straightforward way: for instance, a function of equations is used to equate two functions, and a pair of equations is used to equate two pairs. The benefit of the cubical approach is that this observation, at first purely empirical, is systematized by *defining* equality in every type in terms of the elements of that type in a context extended by a dimension.

Judgmental unicity of equality: boundary separation

In keeping with our desire to provide *convenient* syntax for working with extensional equality, we want proofs $P, Q : \mathsf{Eq}_{i.A}(N_0, N_1)$ of the same equation to be judgmentally equal. Rather than adding a rule to that effect, whose justification in the presence of the elimination rules for equality types would be unclear, we instead impose a more primitive *boundary separation* principle at the judgmental level: every term is completely determined by its boundary.[3]

$$\frac{\Psi \mid r\ dim \qquad \overrightarrow{\Psi, r = \varepsilon \mid \Gamma \vdash M = N : A}}{\Psi \mid \Gamma \vdash M = N : A}$$

In this rule we have abbreviated $\Psi, r = 0 \mid \Gamma \vdash M = N : A$ and $\Psi, r = 1 \mid \Gamma \vdash M = N : A$ as $\overrightarrow{\Psi, r = \varepsilon \mid \Gamma \vdash M = N : A}$. We shall make use of this notation throughout the paper.

they earn the name "equality" rather than "path".

[3] We call this principle "boundary separation" because it turns out to be exactly the fact that the collections of types and elements, when arranged into presheaves on the category of contexts, are *separated* with respect to a certain coverage on this category. We develop this perspective in the extended version of our paper [50].

We can now *derive* a rule that (judgmentally) equates all $P, Q : \mathsf{Eq}_{i.A}(N_0, N_1)$.

Proof. If $P, Q : \mathsf{Eq}_{i.A}(M, N)$, then to show that $P = Q$, it suffices to show that $\lambda i.P(i) = \lambda i.Q(i)$; by the congruence rule for equality abstraction, it suffices to show that $P(i) = Q(i)$ in the extended context. But by boundary separation, we may pivot on the boundary of i, and it suffices to show that $P(0) = Q(0)$ and $P(1) = Q(1)$. But these are automatic, because P and Q are both proofs of $\mathsf{Eq}_{i.A}(N_0, N_1)$, and therefore $P(\varepsilon) = Q(\varepsilon) = N_\varepsilon$. ◀

In an unpublished note from 2017, Thierry Coquand identifies a class of cubical sets equivalent to our separated types, calling them "Bishop sets" [26].

2.1 Kan operations: coercion and composition

How does one *use* a proof of equality? We must have at least a coercion operation which, given a proof $Q : \mathsf{Eq}_{_.\mathcal{U}_k}(A, B)$, coherently transforms elements $M : A$ to elements of B.

2.1.1 Generalized coercion

In XTT, coercion and its coherence are obtained as instances of one general operation: for any two dimensions r, r' and a *line* of types $i.C$, if M is an element of $C\langle r/i\rangle$, then $[i.C]\downarrow^r_{r'} M$ is an element of $C\langle r'/i\rangle$.

$$\frac{\Psi \mid r, r'\ \mathit{dim} \qquad \Psi, i \mid \Gamma \vdash C\ \mathit{type}_k \qquad \Psi \mid \Gamma \vdash M : C\langle r/i\rangle}{\Psi \mid \Gamma \vdash [i.C]\downarrow^r_{r'} M : C\langle r'/i\rangle}$$

In the case of a proof $Q : \mathsf{Eq}_{_.\mathcal{U}_k}(A, B)$ of equality between types, we coerce $M : A$ to the type B using the instance $[i.Q(i)]\downarrow^0_1 M$. But how does M relate to its coercion? *Coherence* of coercion demands their equality, although such an equation must relate terms of (formally) different types; this heterogeneous equality is stated in XTT using a *dependent* equality type $\mathsf{Eq}_{i.Q(i)}(M, [i.Q(i)]\downarrow^0_1 M)$. To construct an element of this equality type, we use the same coercion operator but with a different choice of r, r'; we construct this *filler* by coercing from 0 to a fresh dimension, obtaining $\lambda j.[i.Q(i)]\downarrow^0_j M : \mathsf{Eq}_{i.Q(i)}(M, [i.Q(i)]\downarrow^0_1 M)$:

$$j.Q(j) \quad \ni \qquad M \xrightarrow{\ j.[i.Q(i)]\downarrow^0_j M\ } [i.Q(i)]\downarrow^0_1 M$$

To see that the filler $[i.Q(i)]\downarrow^0_j M$ has the correct boundary with respect to j, we inspect its instances under the substitutions $0/j, 1/j$. First, we observe that the right-hand side $([i.Q(i)]\downarrow^0_j M)\langle 1/j\rangle$ is exactly $[i.Q(i)]\downarrow^0_1 M$; second, we must see that $([i.Q(i)]\downarrow^0_j M)\langle 0/j\rangle$ is M, bringing us to an important equation that we must impose generally:

$$\Psi \mid \Gamma \vdash [i.C]\downarrow^r_r M = M : C\langle r/i\rangle$$

How do coercions compute?

In order to ensure that proofs in XTT can be computed to a canonical form, we need to explain generalized coercion in each type in terms of the elements of that type. To warm up, we explain how coercion must compute in a non-dependent function type:

$$[i.A \to B]\downarrow^r_{r'} M = \lambda x.[i.B]\downarrow^r_{r'} \big(M([i.A]\downarrow^{r'}_r x)\big)$$

That is, we abstract a variable $x : A\langle r'/i\rangle$ and need to obtain an element of type $B\langle r'/i\rangle$. By *reverse* coercion, we obtain $[i.A]\downarrow^{r'}_r x : A\langle r/i\rangle$; by applying M to this, we obtain an

element of type $B\langle r/i\rangle$. Finally, we coerce from r to r'. The version for dependent function types is not much harder, but requires a filler:

$$\frac{\widetilde{x} \triangleq \lambda j.[i.A]\downarrow_j^{r'} x}{[i.(x:A)\to B]\downarrow_{r'}^{r} M = \lambda x.[i.B[\widetilde{x}(i)/x]]\downarrow_{r'}^{r} M(\widetilde{x}(r))}$$

The case for dependent pair types is similar, but without the contravariance:

$$\frac{\widetilde{M_0} \triangleq \lambda j.[i.A]\downarrow_j^{r} \mathsf{fst}(M)}{[i.(x:A)\times B]\downarrow_{r'}^{r} M = \langle \widetilde{M_0}(r'), [i.B[\widetilde{M_0}(i)/x]]\downarrow_{r'}^{r} \mathsf{snd}(M)\rangle}$$

Coercions for base types (like bool) are uniformly determined by *regularity*, a rule of XTT stating that if A is a type which doesn't vary in the dimension i, then $[i.A]\downarrow_{r'}^{r} M$ is just M. Regularity makes type sense because $A\langle r/i\rangle = A = A\langle r'/i\rangle$; semantically, it is more difficult to justify in the presence of standard universes, and is not known to be compatible with principles like univalence.[4] But XTT is specifically designed to provide a theory of *equality* rather than *paths*, so we do not expect or desire to justify univalence at this level.[5]

The only difficult case is to define coercion for equality types; at first, we might try to define $[i.\mathsf{Eq}_{j.A}(N_0,N_1)]\downarrow_{r'}^{r} P$ as $\lambda j.[i.A]\downarrow_{r'}^{r} P(j)$, but this does not make type-sense: we need to see that $\big([i.A]\downarrow_{r'}^{r} P(j)\big)\langle\varepsilon/j\rangle = N_\varepsilon$, but we only obtain $\big([i.A]\downarrow_{r'}^{r} P(j)\big)\langle\varepsilon/j\rangle = [i.A]\downarrow_{r'}^{r} N_\varepsilon$, which is "off by" a coercion. Intuitively, we can solve this problem by specifying what values a coercion takes under certain substitutions: in this case, N_0 under $0/j$, and N_1 under $1/j$. We call the resulting operation *generalized composition*.

2.1.2 Generalized composition

For any dimensions r, r', s and a line of types $i.C$, if M is an element of $C\langle r/i\rangle$ and $i.N_0, i.N_1$ are lines of elements of C defined respectively on the subcubes $(s=0), (s=1)$ such that $N_\varepsilon\langle r/i\rangle = M$, then $[i.C]\downarrow_{r'}^{r} M\ [s \text{ with } 0 \hookrightarrow i.N_0 \mid 1 \hookrightarrow i.N_1]$ is an element of $C\langle r'/i\rangle$. This is called the *composite* of M with N_0, N_1 from r to r', schematically abbreviated $[i.C]\downarrow_{r'}^{r} M\ [s \text{ with } \overrightarrow{\varepsilon \hookrightarrow i.N_\varepsilon}]$. As with coercion, when $r = r'$, we have $[i.C]\downarrow_{r'}^{r} M\ [s \text{ with } \overrightarrow{\varepsilon \hookrightarrow i.N_\varepsilon}] = M$, and moreover, if $s = \varepsilon$, we have $[i.C]\downarrow_{r'}^{r} M\ [s \text{ with } \overrightarrow{\varepsilon \hookrightarrow i.N_\varepsilon}] = N_\varepsilon\langle r'/i\rangle$.

Returning to coercion for equality types, we now have exactly what we need:

$$[i.\mathsf{Eq}_{j.C}(N_0,N_1)]\downarrow_{r'}^{r} P = \lambda j.([i.C]\downarrow_{r'}^{r} P(j)\ [j \text{ with } \overrightarrow{\varepsilon \hookrightarrow _.N_\varepsilon}])$$

Next we must explain how the generalized composition operation computes at each type; in previous works [11], we have seen that it is simpler to instead *define* generalized composition in terms of a simpler *homogeneous* version, in which one composes in a type C rather than a line of types $i.C$; we write $C\downarrow_{r'}^{r} M\ [s \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.N_\varepsilon}]$ for this homogeneous composition, defining the generalized composition in terms of it as follows:

$$[i.C]\downarrow_{r'}^{r} M\ [s \text{ with } \overrightarrow{\varepsilon \hookrightarrow i.N_\varepsilon}] = C\langle r'/i\rangle\downarrow_{r'}^{r} ([i.C]\downarrow_{r'}^{r} M)\ [s \text{ with } \overrightarrow{\varepsilon \hookrightarrow i.[i.C]\downarrow_{r'}^{i} N_\varepsilon}]$$

[4] Regularity is proved by Swan to be incompatible with univalent universes assuming that certain standard techniques are used [53]; however, it is still possible that there is a different way to model univalent universes with regularity. Awodey constructs a model of intensional type theory *without* universes in regular Kan cubical sets [13], using the term *normality* for what we have called regularity.

[5] Indeed, unicity of identity proofs is also incompatible with univalence. XTT is, however, compatible with a formulation in which it is just one level of a two-level type theory, along the lines of Voevodsky's Homotopy Type System, in which the other level would have a univalent notion of path that coexists in harmony with our notion of equality [56, 11].

Surprisingly, in XTT we do not need to build in any computation rules for homogeneous composition, because they are completely determined by judgmental boundary separation. For instance, we can derive a computation rule already for homogeneous composition in the dependent function type, by observing that the equands have the same boundary with respect to the dimension j:

$$(x : A) \to B \downarrow^{r}_{r'} M \; [j \text{ with } \overrightarrow{\varepsilon \hookrightarrow i.N_\varepsilon}] = \lambda x.B \downarrow^{r}_{r'} M(x) \; [j \text{ with } \overrightarrow{\varepsilon \hookrightarrow i.N_\varepsilon}]$$

From homogeneous composition, we obtain *symmetry and transitivity* for the equality types. Given $P : \mathsf{Eq}_{_.A}(M, N)$, we obtain an element of type $\mathsf{Eq}_{_.A}(N, M)$ as follows:

$$\lambda i.A \downarrow^{0}_{1} P(0) \; [i \text{ with } 0 \hookrightarrow j.P(j) \mid 1 \hookrightarrow _.P(0)]$$

Furthermore, given $Q : \mathsf{Eq}_{_.A}(N, O)$, we obtain an element of type $\mathsf{Eq}_{_.A}(M, O)$ as follows:

$$\lambda i.A \downarrow^{0}_{1} P(i) \; [i \text{ with } 0 \hookrightarrow _.P(0) \mid 1 \hookrightarrow j.Q(j)]$$

▶ **Example 2.1** (Identity type)**.** It is possible to *define* Martin-Löf's identity type and its eliminator, albeit with a much stronger computation rule than is customary.

$$\mathsf{Id}_A(M, N) \triangleq \mathsf{Eq}_{_.A}(M, N) \qquad \mathsf{refl}_A(M) \triangleq \lambda _.M$$

$$\frac{\widetilde{P} \triangleq \lambda j.(A \downarrow^{0}_{j} P(0) \; [i \text{ with } 0 \hookrightarrow _.P(0) \mid 1 \hookrightarrow k.P(k)])}{\mathsf{J}_{x,y,p.C(x,y,p)}(P; x.Q(x)) \triangleq [i.C(P(0), P(i), \widetilde{P})] \downarrow^{0}_{1} Q(P(0))}$$

This particular definition of J relies on XTT's *boundary separation* rule, but one could instead define it in a more complicated way without boundary separation. However, that this construction of the identity type models the computation rule relies crucially on *regularity*, which does not hold in other cubical type theories whose path types validate univalence. In the absence of regularity, one can define an operator with the same type as J but which satisfies its computation rule only up to a path.

2.2 Closed universes and type-case

In Section 2.1.1, we showed how to calculate coercions $[i.C] \downarrow^{r}_{r'} M$ in each type former C. In previous cubical type theories [24, 11], one could "uncover" all the things that a coercion must be equal to by reducing according to the rules which inspect the interior of the type line $i.C$. While this strategy can be used to establish canonicity for closed terms, it fails to uncover certain reductions for open terms, a prerequisite for algorithmic type checking.

Specifically, given a variable $q : \mathsf{Eq}_{_.\mathcal{U}_k}(A_0 \to B_0, A_1 \to B_1)$, the coercion $[i.q(i)] \downarrow^{r}_{r'} M$ is *not* necessarily stuck, unlike in other cubical type theories. Suppose that we can find further proofs $Q_A : \mathsf{Eq}_{_.\mathcal{U}_k}(A_0, A_1)$ and $Q_B : \mathsf{Eq}_{_.\mathcal{U}_k}(B_0, B_1)$; in this case, $\lambda i.Q_A(i) \to Q_B(i)$ is *also* a proof of $\mathsf{Eq}_{_.\mathcal{U}_k}(A_0 \to B_0, A_1 \to B_1)$, so by boundary separation it must be equal to q, and therefore $[i.q(i)] \downarrow^{r}_{r'} M$ must be equal to $[i.Q_A(i) \to Q_B(i)] \downarrow^{r}_{r'} M$. But the type-directed reduction rule for coercion applies only to the latter! Generally, to see how to reduce the first coercion, it seems that we need to be able to "dream up" proofs Q_A, Q_B out of thin air, or determine that they can't exist, an impossible task.

In XTT, we cut this Gordian knot by ensuring that Q_A, Q_B *always* exist, following the approach employed in OTT. To invert the equation q into Q_A and Q_B, we add an intensional *type-case* operator to XTT, committing to a closed and inductive notion of universe by allowing pattern-matching on types [46]. It is also possible to extend XTT with open and/or univalent universes which themselves lack boundary separation, as in two-level type theories.

For illustrative purposes, consider coercion along an equality between dependent function types. Given $q : \mathsf{Eq}_{_.\mathcal{U}_k}((x : A_0) \to B_0, (x : A_1) \to B_1)$, we define by type-case the following:

$$Q_A \triangleq \lambda i.\mathsf{tycase}\ q(i)\ [\Pi_A B \mapsto A \mid _ \mapsto \mathsf{bool}] : \mathsf{Eq}_{_.\mathcal{U}_k}(A_0, A_1)$$

$$Q_B \triangleq \lambda i.\mathsf{tycase}\ q(i)\ [\Pi_A B \mapsto B \mid _ \mapsto \lambda_.\mathsf{bool}] : \mathsf{Eq}_{i.Q_A(i) \to \mathcal{U}_k}(\lambda x.B_0, \lambda x.B_1)$$

Because of q's boundary, we are concerned only with the Π branch of the above expressions, and are free to emit a "dummy" answer in other branches. With Q_A, Q_B in hand, we note that $q(i) = (x : Q_A(i)) \to Q_B(i)(x)$ using boundary separation; therefore, we are free to calculate $[i.q(i)] \downarrow^{r}_{r'} M$ as follows:

$$\frac{\widetilde{x} \triangleq \lambda j.[i.Q_A(i)] \downarrow^{r'}_{j} x}{[i.q(i)] \downarrow^{r}_{r'} M = \lambda x.[i.Q_B(i)(\widetilde{x}(i))] \downarrow^{r}_{r'} M(\widetilde{x}(r))}$$

This *lazy* style of computing with proofs of equality means, in particular, that coercing along an equation cannot tell the difference between a postulated axiom and a canonical proof of equality, making XTT compatible with extension by consistent equational axioms.

▶ Remark 2.2. One might wonder whether it is possible to tame the use of type-case above to something compatible with a *parametric* understanding of types, in which (as in OTT) one cannot branch on whether or not C is a function type or a pair type, etc. It is likely that this can be done, but we stress that the fundamental difficulty is not resolved: whether or not we allow general type-case, we have not escaped the need for type constructors to be disjoint and injective, which contradicts the role of universes in mathematics as (weak) classifiers of small families. Future work on XTT and its successors must focus on resolving this issue, quite apart from any considerations of parametricity.

2.3 Future extensions

Universe of propositions

XTT currently lacks one of the hallmarks of OTT, an extensional universe of proof-irrelevant propositions. In future work, we intend to extend XTT with a reflective subuniverse Ω of propositions closed under equality and universal and existential quantification over arbitrary types, satisfying:

- *Proof irrelevance.* For each proposition $\Psi \mid \Gamma \vdash p : \Omega$, we have $\Psi \mid \Gamma \vdash M = N : p$ for all $\Psi \mid \Gamma \vdash M, N : p$.
- *Extensionality (univalence).* For all $\Psi \mid \Gamma \vdash p, q : \Omega$, we have an element of $\mathsf{Eq}_{_.\Omega}(p, q)$ whenever there are functions $p \to q$ and $q \to p$.

The *reflection* of the propositional subuniverse will take a type $\Psi \mid \Gamma \vdash A\ \mathit{type}_k$ to a proof-irrelevant proposition $\Psi \mid \Gamma \vdash \|A\| : \Omega$, acting as a strict truncation or squash type [25, 47, 14]. The addition of Ω will allow XTT to be used as a syntax for topos-theoretic constructions, with Ω playing the role of the subobject classifier.

(Indexed) Quotient Inductive Types

Another natural extension of XTT is the addition of *quotient types*; already considered as an extension to OTT by the Epigram Team [18] and more recently by Atkey [12], quotient types are essential when using type theory for either programming or mathematics. One of the ideas of Homotopy Type Theory and cubical type theories in particular is to reconstruct the notion of quotienting by an equivalence relation as a special case of *higher inductive*

type (HITs), a generalization of ordinary inductive types which allows constructors to target higher dimensions with a specified partial boundary. When working purely at the level of sets, as in XTT, these higher inductive types are called *quotient inductive types* (QITs) [5].

We intend to adapt the work of Cavallo and Harper [22] to a general schema for *indexed quotient inductive types* as an extension of XTT. The resulting system would support ordinary quotients by equivalence relations *en passant*, and when these equivalence relations are valued in Ω, one can show that they are effective. Quotient inductive types also enable the construction of free algebras for infinitary algebraic theories, usually obtained in classical set theory from the non-constructive axiom of choice [17, 39]. Another application of quotient inductive types is the definition of a *localization* functor with respect to a class of maps, enabling users of the extended XTT to work internally with sheaf subtoposes.

The extension of XTT with quotient inductive types means that we must account for *formal homogeneous composites* in QITs which are canonical forms [22, 28]. Ordinarily, this introduces a severe complicating factor to a canonicity proof, because the notion of canonical form ceases to be stable under all dimension substitutions [11, 34], but we expect the *proof-relevant* cubical logical families technique that we introduce in Section 3 to scale directly to the case of quotient inductive types without significant change, in contrast with classical approaches based on partial equivalence relations.

3 Algebraic model theory and canonicity

We have been careful to formulate the XTT language in a *(generalized) algebraic* way, obtaining automatically a category of algebras and homomorphisms which is equipped with an initial object [19, 20, 36]. That this initial object is isomorphic to the model of XTT obtained by constraining and quotienting its raw syntax under judgmental equality (i.e. the Lindenbaum–Tarski algebra) is an instance of Voevodsky's famous Initiality Conjecture [57], and we do not attempt to prove it here; we merely observe that this result has been established for several simpler type theories [51, 21].

Working within the category of XTT-algebras enables us to formulate and prove results like canonicity and normalization for the *initial* XTT-algebra in an economical manner, avoiding the usual bureaucratic overhead of reduction relations and partial equivalence relations, which were the state of the art for type-theoretic metatheory prior to the work of Shulman [48], Altenkirch and Kaposi [4], and Coquand [27].

Because our algebraic techniques involve defining families over *only* well-typed terms already quotiented by judgmental equality, we avoid many of the technical difficulties arising from working with the raw terms of cubical type theories, including the closure under "coherent expansion" which is critical to earlier cubical metatheories [11, 34]. Our abstract gluing-based approach therefore represents a methodological advance in metatheory for cubical type theories.

▶ **Theorem 3.1** (Canonicity). *In the initial XTT-algebra, if* $\cdot \mid \cdot \vdash M : \mathsf{bool}$*, then either* $\cdot \mid \cdot \vdash M = \mathsf{true} : \mathsf{bool}$ *or* $\cdot \mid \cdot \vdash M = \mathsf{false} : \mathsf{bool}$.

Following previous work [49], we employ for our semantics a variant of *categories with families* (cwf) [29] which supports a predicative hierarchy of universes à la Russell. A cwf in our sense begins with a *category of contexts* $\mathcal{C}$, and a presheaf of types $\mathsf{Ty}_{\mathcal{C}} : \widehat{\mathcal{C} \times \mathbb{L}}$; here $\mathbb{L}$ is the category of *universe levels*, with objects the natural numbers and unique arrows $l \longrightarrow k$

if and only if $k \leq l$.[6] The fiber of the presheaf of types $\mathsf{Ty}_{\mathcal{C}} : \widehat{\mathcal{C} \times \mathbb{L}}$ at (Γ, k) is written $\mathsf{Ty}^k_{\mathcal{C}}(\Gamma)$, and contains the types in context Γ of universe level k. Reindexing implements simultaneous substitution $\gamma^* A$ and universe level shifting $\Uparrow^l_k A$. In our metatheory, we assume the Grothendieck Universe Axiom, and consequently obtain a transfinite ordinal-indexed hierarchy of meta-level universes $\mathcal{V}_k$. We impose the requirement that each collection of types $\mathsf{Ty}^k_{\mathcal{C}}(\Gamma)$ is k-small, i.e. $\mathsf{Ty}^k_{\mathcal{C}}(\Gamma) \in \mathcal{V}_k$.

Next, we require a *dependent* presheaf of elements $\mathsf{El}_{\mathcal{C}} : \widehat{\int \mathsf{Ty}_{\mathcal{C}}}$, whose fibers $\mathsf{El}_{\mathcal{C}}((\Gamma, k), A)$ we write $\mathsf{El}_{\mathcal{C}}(\Gamma \vdash A)$; to interpret the actions of level lifting on terms properly, we require the functorial actions $\mathsf{El}_{\mathcal{C}}(\Gamma \vdash A) \longrightarrow \mathsf{El}_{\mathcal{C}}(\Gamma \vdash \Uparrow^l_k A)$ to be identities, strictly equating the fibers $\mathsf{El}_{\mathcal{C}}(\Gamma \vdash A)$ and $\mathsf{El}_{\mathcal{C}}(\Gamma \vdash \Uparrow^l_k A)$. The remaining data of a basic cwf is a *context comprehension*, which for every context Γ and type $A \in \mathsf{Ty}^k_{\mathcal{C}}(\Gamma)$ determines an extended context $\Gamma.A$ with a weakening substitution $\Gamma.A \xrightarrow{\mathbf{p}} \Gamma$ and a variable term $\mathbf{q} \in \mathsf{El}_{\mathcal{C}}(\Gamma.A \vdash \mathbf{p}^* A)$.

Next, we specify what further structure is required to make such a cwf into an XTT-algebra. To represent contexts Ψ semantically, we use the *augmented Cartesian cube category* $\square_+$, which adjoins to the Cartesian cube category $\square$ an initial object; from this, we obtain equalizers $0 = 1$ in addition to the equalizers $i = r$ which exist in $\square$. We then require a *split fibration* $\mathcal{C} \xrightarrow{\mathfrak{u}} \square_+$ with a terminal object, which implements the dependency of contexts Γ on cubes Ψ and forces appropriate dimension restrictions to exist for contexts, types and elements. The split fibration induces all the structure necessary to implement dimension operations; we refer the reader to the extended version of our paper [50] for details. In the following discussion, we limit ourselves to a few simpler consequences. First, we can apply dimension substitutions in terms and types, writing $\psi^\ddagger_\Gamma A$ to apply ψ in a type A in context Γ. We can also apply dimension substitutions to contexts, written $\psi^*\Gamma$. We write $\hat{\imath}$ for the dimension substitution which weakens by a dimension variable i. Finally, we write $\mathsf{Dim}_{\mathcal{C}}(\Gamma)$ for the set of valid dimensions expressions generated from $\mathfrak{u}(\Gamma)$.

▶ **Requirement** (Boundary separation in models)**.** In order to enforce boundary separation in XTT-algebras we require that types and elements over them satisfy a separation property. In the extended version of our paper [50] we phrase the full condition as a separation requirement with respect to a particular Grothendieck topology on the category of contexts. A specific consequence is the familiar boundary separation principle for types: given two types $A, B \in \mathsf{Ty}_{\mathcal{C}}(\Gamma)$ and a dimension $i \in \mathfrak{u}(\Gamma)$, if $(\epsilon/i)^\ddagger A = (\epsilon/i)^\ddagger B$ for each $\epsilon \in \{0, 1\}$ then $A = B$.

▶ **Requirement** (Coercion in models)**.** An XTT-algebra must also come with a *coercion structure*, specifying how generalized coercion is interpreted in each type. For every type $A \in \mathsf{Ty}^n_{\mathcal{C}}(\hat{\imath}^*\Gamma)$ over Ψ, i, dimensions $r, r' \in \mathsf{Dim}_{\mathcal{C}}(\Gamma)$, and element $M \in \mathsf{El}_{\mathcal{C}}(\Gamma \vdash (r/i)^\ddagger_{\hat{\imath}^*\Gamma} A)$, we require an element $\mathbf{coe}^{r \rightsquigarrow r'}_{i.A}\, M \in \mathsf{El}_{\mathcal{C}}(\Gamma \vdash (r'/i)^\ddagger_{\hat{\imath}^*\Gamma} A)$ with the following properties (in addition to naturality requirements):

- *Adjacency.* If $r = r'$ then $\mathbf{coe}^{r \rightsquigarrow r'}_{i.A}\, M = M$.
- *Regularity.* If $A = \hat{\imath}^\ddagger_\Gamma A'$ for some $A' \in \mathsf{Ty}^n_{\mathcal{C}}(\Gamma)$, then $\mathbf{coe}^{r \rightsquigarrow r'}_{i.A}\, M = M$.

Additional equations in later requirements specify that generalized coercion computes properly in each connective. Similarly, a model must be equipped with a *composition structure* which specifies the interpretation of the composition operator.

Finally, we specify algebraically the data with which such a cwf must be equipped in order to model all the connectives of XTT (again, details are contained in the extended

[6] Observe that $\mathbb{L} = \omega^{\mathsf{op}}$; reversing arrows allows us to move types from smaller universes to larger ones.

version of our paper [50]); to distinguish the abstract (De Bruijn) syntax of the cwf from the raw syntax of XTT we use boldface, writing $\mathbf{\Pi}(A, B)$, $\mathbf{papp}(i.A, M, r)$ and $\mathbf{U}_k$ to correspond to $(x : A) \to B$, $\mathsf{app}_{i.A}(M, r)$ and $\mathcal{U}_k$ respectively, etc. We take a moment to specify how some of the primitives of XTT are translated into requirements on a model.

▶ **Requirement** (Dependent equality types in models)**.** An XTT-algebra must model *dependent equality types*, which is to say that the following structure is exhibited:

- *Formation.* For each type $A \in \mathsf{Ty}^n_{\mathcal{C}}(\hat{\imath}^*\Gamma)$ and elements $\overrightarrow{N_\varepsilon \in \mathsf{El}_{\mathcal{C}}(\Gamma \vdash (\varepsilon/i)^\ddagger A)}$, a type $\mathbf{Eq}(i.A, N_0, N_1) \in \mathsf{Ty}^n_{\mathcal{C}}(\Gamma)$.
- *Introduction.* For each $M \in \mathsf{El}_{\mathcal{C}}(\hat{\imath}^*\Gamma \vdash A)$, an element $\mathbf{plam}(i.A, M) \in \mathsf{El}_{\mathcal{C}}(\Gamma \vdash \mathbf{Eq}(i.A, (0/i)^\ddagger M, (1/i)^\ddagger M))$.
- *Elimination.* For each $M \in \mathsf{El}_{\mathcal{C}}(\Gamma \vdash \mathbf{Eq}(i.A, N_0, N_1))$ and $r \in \mathsf{Dim}_{\mathcal{C}}(\Gamma)$, an element $\mathbf{papp}(i.A, M, r) \in \mathsf{El}_{\mathcal{C}}(\Gamma \vdash (r/i)^\ddagger A)$ satisfying the equations $\overrightarrow{\mathbf{papp}(i.A, M, \epsilon) = N_\epsilon}$.
- *Computation.* For $M \in \mathsf{El}_{\mathcal{C}}(\hat{\imath}^*\Gamma \vdash A)$ and $r \in \mathsf{Dim}_{\mathcal{C}}(\Gamma)$, the equation:

$$\mathbf{papp}(i.A, \mathbf{plam}(i.A, i.M), r) = (r/i)^\ddagger M$$

- *Unicity.* For $M \in \mathsf{El}_{\mathcal{C}}(\Gamma \vdash \mathbf{Eq}(i.A, N_0, N_1))$, $M = \mathbf{plam}(i.A, j.\mathbf{papp}(i.\hat{\jmath}^\ddagger A, \hat{\jmath}^\ddagger M, j))$.
- *Level restriction.* The following equations:

$$\Uparrow^l_k \mathbf{Eq}(i.A, N_0, N_1) = \mathbf{Eq}(i.\Uparrow^l_k A, N_0, N_1) \qquad \mathbf{plam}(i.\Uparrow^l_k A, M)r = \mathbf{plam}(i.A, M)r$$

$$\mathbf{papp}(i.\Uparrow^l_k A, M, r) = \mathbf{papp}(i.A, M, r)$$

- *Naturality.* For $\Delta \xrightarrow{\gamma} \Gamma$, the following naturality equations:

$$\gamma^*\mathbf{Eq}(i.A, N_0, N_1) = \mathbf{Eq}(i.(\hat{\imath}^+\gamma)^* A, \gamma^* N_0, \gamma^* N_1)$$

$$\gamma^*\mathbf{plam}(i.A, i.M) = \mathbf{plam}(i.(\hat{\imath}^+\gamma)^* A, i.(\hat{\imath}^+\gamma)^* M)$$

$$\gamma^*\mathbf{papp}(i.A, M, r) = \mathbf{papp}(i.(\hat{\imath}^+\gamma)^* A, \gamma^* M, \gamma^* r)$$

- *Coercion.* When $\Gamma \overset{\mathfrak{u}}{\longmapsto} \Psi, j$ and $M \in \mathsf{El}_{\mathcal{C}}((r/j)^*\Gamma \vdash (r/j)^\ddagger \mathbf{Eq}(i.A, N_0, N_1))$ where $\Psi \mid r, r'$ *dim*, we require that $\mathbf{coe}^{r \rightsquigarrow r'}_{j.\mathbf{Eq}(i.A, N_0, N_1)}\ M$ equals the following abstraction:

$$\mathbf{plam}(i.(r'/j)^\ddagger A, i.\mathbf{com}^{r \rightsquigarrow r'}_{j.A}\ \mathbf{papp}(i.(r/j)^\ddagger A, \hat{\imath}^\ddagger M, i)\ [i \mathsf{\ with\ } \overrightarrow{\varepsilon \hookrightarrow j.\hat{\jmath}^\ddagger N_\varepsilon}])$$

Any model of extensional type theory can be used to construct a model of XTT, so long as it is equipped with a cumulative, inductively defined hierarchy of universes closed under dependent function types, dependent pair types, extensional equality types and booleans. (Meaning explanations in the style of Martin-Löf [43] are one such model.) The interpretation of XTT into extensional models involves erasing dimensions, coercions, and compositions; the only subtlety, easily managed, is to ensure that all judgments under absurd constraints hold.

3.1 The cubical logical families construction

Any XTT-algebra $\mathcal{C}$ extends to a category $\mathcal{C}^\star$ of *proof-relevant logical predicates*, which we call *logical families* by analogy. The proof-relevant character of the construction enables a simpler proof of canonicity than is obtained with proof-irrelevant techniques, such as partial equivalence relations. Logical families are a type-theoretic version of the *categorical gluing* construction, in which a very rich semantic category (such as sets) is cut down to include

just the morphisms which track definable morphisms in $\mathcal{C}$;[7] one then uses the rich structure of the semantic category to obtain metatheoretic results about syntax (choosing $\mathcal{C}$ to be the initial model) without considering raw terms at any point in the process.

Usually, to prove canonicity one glues the initial model $\mathcal{C}$ together with **Set** along the global sections functor; this equips each context Γ with a family of sets $\Gamma^\bullet$ indexed in the *closing substitutions* for Γ. In order to prove canonicity for a cubical language like XTT, we will need a more sophisticated version of this construction, in which the global sections functor is replaced with something that determines substitutions which are closed with respect to term variables, but open with respect to dimension variables.

The split fibration $\mathcal{C} \xrightarrow{\mathsf{u}} \square_+$ induces a functor $\square_+ \xrightarrow{\langle - \rangle} \mathcal{C}$ which takes every cube Ψ to the empty variable context over Ψ. This functor in turn induces a *nerve* construction $\mathcal{C} \xrightarrow{(\!|-|\!)} \widehat{\square}_+$, taking Γ to the cubical set $\mathcal{C}(\langle - \rangle, \Gamma)$.[8] Intuitively, this is the presheaf of substitutions which are closed with respect to term variables, but open with respect to dimension variables; when wearing $\widehat{\square}_+$-tinted glasses, these appear to be the closed substitutions.

This nerve construction extends to the presheaves of types and elements; we define the fiber of $(\!|\mathsf{Ty}_k|\!) : \widehat{\square}_+$ at Ψ to be the set $\mathsf{Ty}^k_{\mathcal{C}}(\langle\Psi\rangle)$; likewise, we define the fiber of $(\!|\mathsf{El}_k|\!) : \widehat{\int (\!|\mathsf{Ty}_k|\!)}$ at (Ψ, A) to be the set $\mathsf{El}_{\mathcal{C}}(\langle\Psi\rangle \vdash A)$. Internally to $\widehat{\square}_+$, we regard $(\!|\mathsf{El}_k|\!)$ as a dependent type over $(\!|\mathsf{Ty}_k|\!)$. We will then (abusively) write $(\!|A|\!)$ for the fiber of $(\!|\mathsf{El}_k|\!)$ determined by $A : (\!|\mathsf{Ty}_k|\!)$.

Category of cubical logical families

Gluing $\mathcal{C}$ together with $\widehat{\square}_+$ along $(\!|-|\!)$ gives us a category of *cubical logical families* $\mathcal{C}^\star$ whose objects are pairs $\overline{\Gamma} = (\Gamma, \Gamma^\bullet)$, with $\Gamma : \mathcal{C}$ and $\Gamma^\bullet$ a *dependent cubical set* over the cubical set $(\!|\Gamma|\!)$. In other words, $\Gamma^\bullet$ is a "Kripke logical family" on the substitutions $\langle\Psi\rangle \longrightarrow \Gamma$ which commutes with dimension substitutions $\Psi' \longrightarrow \Psi$. A morphism $\overline{\Delta} \longrightarrow \overline{\Gamma}$ is a substitution $\Delta \xrightarrow{\gamma} \Gamma$ together with a proof that γ preserves the logical family: that is, a closed element $\gamma^\bullet$ of the type $\prod_{\delta:(\!|\Delta|\!)} \Delta^\bullet(\delta) \to \Gamma^\bullet(\gamma^*\delta)$ in the internal type theory of $\widehat{\square}_+$. We write $\overline{\gamma}$ for the pair $(\gamma, \gamma^\bullet)$. We have a fibration $\mathcal{C}^\star \xrightarrow{\pi_{\mathsf{syn}}} \mathcal{C}$ which merely projects Γ from $\overline{\Gamma} = (\Gamma, \Gamma^\bullet)$.

Glued type structure

Recall from Section 2.2 that we must model *closed* universes. Therefore, the standard presheaf universes which lift $\mathcal{V}_k$ to (weakly) classify all k-small presheaves are insufficient in our case; instead, we must equip each type with a *code* so that type-case is definable. Accordingly, we define for each $n \in \mathbb{N}$ an inductive cubical set $\mathfrak{U}^\bullet_n A : \mathcal{V}_{n+1}$ indexed over $A : (\!|\mathsf{Ty}_n|\!)$; internally to $\widehat{\square}_+$, the cubical set $\mathfrak{U}^\bullet_n A$ is the collection of *realizers* for the $\mathcal{C}$-type A. An imprecise but helpful analogy is to think of a realizer $\mathtt{A} : \mathfrak{U}^\bullet_n A$ as something like a whnf of A, with the caveat that $\mathtt{A}$ is an element of this inductively defined set, not a $\mathcal{C}$-type. Simultaneously, for each $\mathtt{A} : \mathfrak{U}^\bullet_n A$, we define a cubical family $\mathtt{A}^\circ : (\!|A|\!) \to \mathcal{V}_n$ of realizers of elements of A, with each $\mathtt{A}^\circ$ being the *logical family* of the $\mathcal{C}$-type A; finally, we also define realizers for coercion and composition by recursion on the realizers for types.[9] A fragment of this definition is

[7] The gluing construction is similar to realizability; the main difference is that in gluing, one considers collections of "realizers" which are *not* all drawn from a single computational domain.

[8] This construction is also called the *relative hom functor* by Fiore [30]; its use in logic originates in the study of definability for λ-calculus, characterizing the domains of discourse for Kripke logical predicates of varying arity [35]. We learned the connection to the abstract nerve construction in conversations with M. Fiore about his unpublished joint work with S. Awodey.

[9] It is important to note that we do *not* use large induction-recursion in $\widehat{\square}_+$ (to our knowledge, the construction of inductive-recursive definitions has not yet been lifted to presheaf toposes); instead, we

$$\frac{(j < n)}{\mathtt{univ}_j : \mathfrak{U}^\bullet_n \mathbf{U}_j} \qquad \frac{}{\mathtt{bool} : \mathfrak{U}^\bullet_n \mathbf{bool}}$$

$$\frac{\mathtt{A} : \mathfrak{U}^\bullet_n A \qquad \mathtt{B} : \prod_{M:(\!|A|\!)} \mathtt{A}^\circ M \to \mathfrak{U}^\bullet_n(\langle \mathbf{id}, M\rangle^* B)}{\mathtt{pi}(\mathtt{A};\mathtt{B}) : \mathfrak{U}^\bullet_n \mathbf{\Pi}(A,B) \qquad \mathtt{sg}(\mathtt{A};\mathtt{B}) : \mathfrak{U}^\bullet_n \mathbf{\Sigma}(A,B)} \qquad \frac{\mathtt{A} : \prod_{i:\mathbb{I}} \mathfrak{U}^\bullet_n A_i \qquad \overrightarrow{N^\bullet_\varepsilon : \mathtt{A}(\varepsilon)^\circ N_\varepsilon}}{\mathtt{eq}(\mathtt{A}; N^\bullet_0, N^\bullet_1) : \mathfrak{U}^\bullet_n \mathbf{Eq}(i.A_i, N_0, N_1)}$$

$$\begin{aligned}
\mathtt{univ}^\circ_n A &= \mathfrak{U}^\bullet_n A \\
\mathtt{bool}^\circ M &= (M = \mathbf{true}) + (M = \mathbf{false}) \\
\mathtt{pi}(\mathtt{A};\mathtt{B})^\circ M &= \textstyle\prod_{N:(\!|A|\!)} \prod_{N^\bullet:\mathtt{A}^\bullet N} (\mathtt{B}NN^\bullet)^\circ \mathbf{app}(A, B, M, N) \\
\mathtt{sg}(\mathtt{A};\mathtt{B})^\circ M &= \textstyle\sum_{M^\bullet_0 : \mathtt{A}^\circ \mathbf{fst}(A,B,M)} (\mathtt{B}(\mathbf{fst}(A,B,M)) M^\bullet_0)^\circ \mathbf{snd}(A,B,M) \\
\mathtt{eq}(\mathtt{A}; N^\bullet_0, N^\bullet_1)^\circ M &= \left\{ M^\bullet : \textstyle\prod_{i:\mathbb{I}} \mathtt{A}(i)^\circ \mathbf{papp}(i.A, M, i) \mid \overrightarrow{M^\bullet(\varepsilon) = N^\bullet_\varepsilon} \right\}
\end{aligned}$$

$$\begin{aligned}
[i.\mathtt{bool}] \downarrow^r_{r'} M^\bullet &= M^\bullet \\
[i.\mathtt{pi}(\mathtt{A};\mathtt{B})] \downarrow^r_{r'} M^\bullet &= \lambda N^\bullet.\, [i.\mathtt{B}([i.\mathtt{A}] \downarrow^{r'}_i N^\bullet)] \downarrow^r_{r'} M^\bullet \big([i.\mathtt{A}] \downarrow^{r'}_r N^\bullet\big) \\
[i.\mathtt{eq}(\mathtt{A}; N^\bullet_0, N^\bullet_1)] \downarrow^r_{r'} M^\bullet &= \lambda k.\, [i.\mathtt{A}k] \downarrow^r_{r'} M^\bullet k\ [k \text{ with } \overrightarrow{\varepsilon \hookrightarrow _.N^\bullet_\varepsilon}] \\
\mathtt{pi}(\mathtt{A};\mathtt{B}) \downarrow^r_{r'} M^\bullet\ [s \text{ with } \overrightarrow{\varepsilon \hookrightarrow i.M'^\bullet i}] &= \lambda N^\bullet.\, \mathtt{B}N^\bullet \downarrow^r_{r'} M^\bullet N^\bullet\ [s \text{ with } \overrightarrow{\varepsilon \hookrightarrow i.M'^\bullet i N N^\bullet}] \\
\mathtt{eq}(\mathtt{A}; N^\bullet_0, N^\bullet_1) \downarrow^r_{r'} M^\bullet\ [s \text{ with } \overrightarrow{\varepsilon \hookrightarrow i.M'^\bullet i}] &= \lambda j.\, \mathtt{A}j \downarrow^r_{r'} M^\bullet j\ [s \text{ with } \overrightarrow{\varepsilon \hookrightarrow i.M'^\bullet ij}] \\
&\vdots
\end{aligned}$$

Figure 2 The inductive definition of realizers $\mathfrak{U}^\bullet_n A : \mathcal{V}_{n+1}$ for types $A : (\!|\mathsf{Ty}_n|\!)$ in $\widehat{\square}_+$; we also include a fragment of the realizers for Kan operations, which are also defined by recursion on the realizers for types. We write $\mathbb{I}$ for the representable presheaf $\mathbf{y}(i)$.

summarized in Figure 2. In the definition of $\mathtt{A}^\circ$ we freely make use of the internal type theory of $\widehat{\square}_+$. This not only exposes the underlying *logical relations* flavor of these definitions but simplifies a number of proofs (see the extended version of our paper [50]).

From all this, we can define the cwf structure on $\mathcal{C}^\star$. We obtain a presheaf of types $\mathsf{Ty}_{\mathcal{C}^\star} : \widehat{\mathcal{C}^\star \times \mathbb{L}}$ by taking $\mathsf{Ty}^k_{\mathcal{C}^\star}(\overline{\Gamma})$ to be the set of pairs $\overline{A} = (A, A^\bullet)$ where $A \in \mathsf{Ty}^k_{\mathcal{C}}(\Gamma)$ and $A^\bullet$ is an element of the type $\prod_{\gamma:(\!|\Gamma|\!)} \prod_{\gamma^\bullet:\Gamma^\bullet(\gamma)} \mathfrak{U}^\bullet_k(\gamma^* A)$ in the internal type theory of $\widehat{\square}_+$. To define the dependent presheaf of elements, we take $\mathsf{El}_{\mathcal{C}^\star}(\overline{\Gamma} \vdash \overline{A})$ to be the set of pairs $\overline{M} = (M, M^\bullet)$ where $M \in \mathsf{El}_{\mathcal{C}}(\Gamma \vdash A)$ and $M^\bullet$ is an element of the type $\prod_{\gamma:(\!|\Gamma|\!)} \prod_{\gamma^\bullet:\Gamma^\bullet(\gamma)} (A^\bullet \gamma\gamma^\bullet)^\circ(\gamma^* M)$ in the internal type theory of $\widehat{\square}_+$. In this model, the context comprehension operation $\overline{\Gamma}.\overline{A}$ is defined as the pair $(\Gamma.A, (\overline{\Gamma}.\overline{A})^\bullet)$ where $(\overline{\Gamma}.\overline{A})^\bullet\langle\gamma, M\rangle$ is the cubical set $\sum_{\gamma^\bullet:\Gamma^\bullet(\gamma)} (A^\bullet\gamma\gamma^\bullet)^\circ(\gamma^* M)$; it is easy to see that we obtain realizers for the weakening substitution and the variable term.

▶ **Construction 3.2** (Dependent equality types in $\mathcal{C}^\star$). Recall that we required a model of

model n object universes using the meta-universe $\mathcal{V}_{n+1}$. This is an instance of *small induction-recursion*, which can be translated into indexed inductive definitions which exist in every presheaf topos [31, 45].

XTT to have sufficient structure to interpret dependent equality types. Here, we discuss how to obtain the formation rule; the full construction can be found in the extended version of our paper [50]. Suppose $\overline{A} \in \mathsf{Ty}^n_{\mathcal{C}^\star}(\hat{\imath}^*\overline{\Gamma})$ and elements $\overline{N_0}$ and $\overline{N_1}$ with $\overline{N}_\varepsilon \in \mathsf{El}_{\mathcal{C}^\star}(\overline{\Gamma} \vdash (\varepsilon/i)^\ddagger\overline{A})$. We wish to construct a type in $\mathsf{Ty}^n_{\mathcal{C}^\star}(\hat{\imath}^*\overline{\Gamma})$.

In $\mathcal{C}^\star$, such a type is a pair of a type $E \in \mathsf{Ty}^n_{\mathcal{C}}(\hat{\imath}^*\Gamma)$ from $\mathcal{C}$ with an element witnessing the logical family $\prod_{\gamma:(\!|\Gamma|\!)}\prod_{\gamma^\bullet:\Gamma^\bullet(\gamma)}\mathfrak{U}^\bullet_k(\gamma^*E)$. We will set the first component to the dependent equality type from $\mathcal{C}$ itself, namely $E = \mathbf{Eq}(i.A, N_0, N_1)$. For the second component, we wish to construct an element of $\prod_{\gamma:(\!|\Gamma|\!)}\prod_{\gamma^\bullet:\Gamma^\bullet(\gamma)}\mathfrak{U}^\bullet_k(\gamma^*\mathbf{Eq}(i.A, N_0, N_1))$. Inspecting the rules for $\mathfrak{U}^\bullet_k$ from Figure 2, there is only one choice: $E^\bullet = \lambda\gamma.\lambda\gamma^\bullet.\mathsf{eq}(A^\bullet\gamma\gamma^\bullet; N_0^\bullet\gamma\gamma^\bullet, N_1^\bullet\gamma\gamma^\bullet)$.

▶ **Construction 3.3** (Coercion in $\mathcal{C}^\star$)**.** The coercion structure on $\mathcal{C}^\star$ is constructed from the coercion structures on $\mathcal{C}$ and the coercion operator for codes from Figure 2.

Given a type $\overline{A} \in \mathsf{Ty}^n_{\overline{\Gamma}}(\hat{\imath}^*\overline{\Gamma})$ over Ψ, i, dimensions $r, r' \in \mathsf{Dim}_{\mathcal{C}}(\overline{\Gamma})$, and an element $\overline{M} \in \mathsf{El}_{\mathcal{C}}(\overline{\Gamma} \vdash (r/i)^\ddagger_{\hat{\imath}^*\overline{\Gamma}}\overline{A})$, we must construct an element of $\mathsf{El}_{\mathcal{C}}(\overline{\Gamma} \vdash (r'/i)^\ddagger_{\hat{\imath}^*\overline{\Gamma}}\overline{A})$. This element must be a pair of $N \in \mathsf{El}_{\mathcal{C}}(\Gamma \vdash (r'/i)^\ddagger_{\hat{\imath}^*\Gamma}A)$ and a term $N^\bullet : \prod_{\gamma:(\!|\Gamma|\!)}\prod_{\gamma^\bullet:\Gamma^\bullet(\gamma)}(A^\bullet\gamma\gamma^\bullet)^\circ(\gamma^*N)$. For the former, we rely on the coercion structure for $\mathcal{C}$ and pick $N = \mathbf{coe}^{r\rightsquigarrow r'}_{i.A}\,M$. For the latter, we use the coercion operation on codes defined in Figure 2 and choose $N^\bullet = \lambda\gamma.\lambda\gamma^\bullet.[i.A^\bullet\gamma\gamma^\bullet]\downarrow^r_{r'} M^\bullet\gamma\gamma^\bullet$.

It is routine to check that this coercion structure enjoys adjacency, regularity, and naturality once the corresponding properties are checked for the coercion operator on codes.

▶ **Theorem 3.4.** *$\mathcal{C}^\star$ is an XTT-algebra, and moreover, $\mathcal{C}^\star \xrightarrow{\pi_{\mathsf{syn}}} \mathcal{C}$ is a homomorphism of XTT-algebras.*

3.2 Canonicity theorem

Because $\mathcal{C}^\star$ is an XTT-algebra, we are now equipped to prove a canonicity theorem for the initial XTT-algebra $\mathcal{C}$: if M is an element of type **bool** in the empty context, then either $M = \mathbf{true}$ or $M = \mathbf{false}$, and not both.

Proof. We have $M \in \mathsf{El}_{\mathcal{C}}(\cdot \vdash \mathbf{bool})$, and therefore $[\![M]\!] \in \mathsf{El}_{\mathcal{C}^\star}(\cdot \vdash \overline{\mathbf{bool}})$. From this we obtain $N : \mathsf{El}_{\mathcal{C}}(\cdot \vdash \mathbf{bool})$ where $N = \pi_{\mathsf{syn}}[\![M]\!]$, and $N^\bullet \in \mathtt{bool}^\circ_\cdot(N)$; by definition, $N^\bullet$ is either a proof that $N = \mathbf{true}$ or a proof that $N = \mathbf{false}$ (see Figure 2). Therefore, it suffices to observe that $\pi_{\mathsf{syn}}[\![M]\!] = M$; but this follows from the universal property of the initial XTT-algebra and the fact that $\mathcal{C}^\star \xrightarrow{\pi_{\mathsf{syn}}} \mathcal{C}$ is an XTT-homomorphism. Moreover, because the interpretation of **bool** in $\mathcal{C}^\star$ is disjoint, M cannot equal both **true** and **false**. ◀

References

1 Andreas Abel, Thierry Coquand, and Peter Dybjer. On the Algebraic Foundation of Proof Assistants for Intuitionistic Type Theory. In Jacques Garrigue and Manuel V. Hermenegildo, editors, *Functional and Logic Programming*, pages 3–13. Springer Berlin Heidelberg, 2008.

2 Stuart Frazier Allen. *A non-type-theoretic semantics for type-theoretic language*. phdthesis, Cornell University, 1987.

3 Thorsten Altenkirch. Extensional equality in intensional type theory. In *Proceedings. 14th Symposium on Logic in Computer Science (Cat. No. PR00158)*, pages 412–420, July 1999. doi:10.1109/LICS.1999.782636.

4 Thorsten Altenkirch and Ambrus Kaposi. Normalisation by Evaluation for Dependent Types. In Delia Kesner and Brigitte Pientka, editors, *1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016)*, volume 52 of *Leibniz International Proceedings*

in Informatics (LIPIcs), pages 6:1–6:16. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.FSCD.2016.6.

5 Thorsten Altenkirch and Ambrus Kaposi. Type Theory in Type Theory Using Quotient Inductive Types. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '16, pages 18–29. ACM, 2016. doi:10.1145/2837614.2837638.

6 Thorsten Altenkirch and Conor McBride. Towards Observational Type Theory, 2006. URL: www.strictlypositive.org/ott.pdf.

7 Thorsten Altenkirch, Conor McBride, and Wouter Swierstra. Observational Equality, Now! In *Proceedings of the 2007 Workshop on Programming Languages Meets Program Verification*, PLPV '07, pages 57–68. ACM, 2007.

8 Carlo Angiuli, Guillaume Brunerie, Thierry Coquand, Kuen-Bang Hou (Favonia), Robert Harper, and Daniel R. Licata. Syntax and Models of Cartesian Cubical Type Theory, February 2019. Preprint. URL: https://github.com/dlicata335/cart-cube.

9 Carlo Angiuli, Evan Cavallo, Kuen-Bang Hou (Favonia), Robert Harper, Anders Mörtberg, and Jonathan Sterling. redtt: implementing Cartesian cubical type theory. Dagstuhl Seminar 18341: Formalization of Mathematics in Type Theory. URL: http://www.jonmsterling.com/pdfs/dagstuhl.pdf.

10 Carlo Angiuli, Evan Cavallo, Kuen-Bang Hou (Favonia), Robert Harper, and Jonathan Sterling. The RedPRL Proof Assistant (Invited Paper). In *Proceedings of the 13th International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice, LFMTP@FSCD 2018, Oxford, UK, 7th July 2018.*, pages 1–10, 2018. doi:10.4204/EPTCS.274.1.

11 Carlo Angiuli, Kuen-Bang Hou (Favonia), and Robert Harper. Cartesian Cubical Computational Type Theory: Constructive Reasoning with Paths and Equalities. In Dan Ghica and Achim Jung, editors, *27th EACSL Annual Conference on Computer Science Logic (CSL 2018)*, volume 119 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:17. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.CSL.2018.6.

12 Robert Atkey. Simplified Observational Type Theory, 2018. URL: https://github.com/bobatkey/sott.

13 Steve Awodey. A cubical model of homotopy type theory. *Annals of Pure and Applied Logic*, 169(12):1270–1294, 2018. Logic Colloquium 2015. doi:10.1016/j.apal.2018.08.002.

14 Steven Awodey and Andrej Bauer. Propositions As [Types]. *J. Log. and Comput.*, 14(4):447–471, August 2004. doi:10.1093/logcom/14.4.447.

15 Andrej Bauer, Gaëtan Gilbert, Philipp Haselwarter, Matija Pretnar, and Christopher A. Stone. Design and Implementation of the Andromeda proof assistant, 2016. TYPES. URL: http://www.types2016.uns.ac.rs/images/abstracts/bauer2.pdf.

16 Errett Bishop. *Foundations of Constructive Analysis*. McGraw-Hill, 1967.

17 Andreas Blass. Words, free algebras, and coequalizers. *Fundamenta Mathematicae*, 117(2):117–160, 1983. URL: http://eudml.org/doc/211359.

18 Edwin Brady, James Chapman, Pierre-Évariste Dagand, Adam Gundry, Conor McBride, Peter Morris, Ulf Norell, and Nicolas Oury. An Epigram Implementation, February 2011.

19 John Cartmell. *Generalised Algebraic Theories and Contextual Categories*. phdthesis, Oxford University, January 1978.

20 John Cartmell. Generalised algebraic theories and contextual categories. *Annals of Pure and Applied Logic*, 32:209–243, 1986.

21 Simon Castellan, Pierre Clairambault, and Peter Dybjer. Undecidability of Equality in the Free Locally Cartesian Closed Category (Extended version). *Logical Methods in Computer Science*, 13(4), 2017.

22 Evan Cavallo and Robert Harper. Higher Inductive Types in Cubical Computational Type Theory. *Proc. ACM Program. Lang.*, 3(POPL):1:1–1:27, January 2019. doi:10.1145/3290314.

23 James Chapman, Fredrik Nordvall Forsberg, and Conor McBride. The Box of Delights (Cubical Observational Type Theory), January 2018. Unpublished note. URL: https://github.com/msp-strath/platypus/blob/master/January18/doc/CubicalOTT/CubicalOTT.pdf.

24 Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical Type Theory: a constructive interpretation of the univalence axiom. *IfCoLog Journal of Logics and their Applications*, 4(10):3127–3169, November 2017. URL: http://www.collegepublications.co.uk/journals/ifcolog/?00019.

25 R. L. Constable, S. F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cremer, R. W. Harper, D. J. Howe, T. B. Knoblock, N. P. Mendler, P. Panangaden, J. T. Sasaki, and S. F. Smith. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice-Hall, Inc., 1986.

26 Thierry Coquand. Universe of Bishop sets, February 2017. URL: http://www.cse.chalmers.se/~coquand/bishop.pdf.

27 Thierry Coquand. Canonicity and normalization for Dependent Type Theory, October 2018. arXiv:1810.09367.

28 Thierry Coquand, Simon Huber, and Anders Mörtberg. On Higher Inductive Types in Cubical Type Theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, pages 255–264. ACM, 2018. doi:10.1145/3209108.3209197.

29 Peter Dybjer. Internal type theory. In Stefano Berardi and Mario Coppo, editors, *Types for Proofs and Programs: International Workshop, TYPES '95 Torino, Italy, June 5–8, 1995 Selected Papers*, pages 120–134. Springer Berlin Heidelberg, 1996.

30 Marcelo Fiore. Semantic Analysis of Normalisation by Evaluation for Typed Lambda Calculus. In *Proceedings of the 4th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming*, PPDP '02, pages 26–37. ACM, 2002. doi:10.1145/571157.571161.

31 Peter Hancock, Conor McBride, Neil Ghani, Lorenzo Malatesta, and Thorsten Altenkirch. Small Induction Recursion. In Masahito Hasegawa, editor, *Typed Lambda Calculi and Applications: 11th International Conference, TLCA 2013, Eindhoven, The Netherlands, June 26-28, 2013. Proceedings*, pages 156–172. Springer Berlin Heidelberg, 2013.

32 Martin Hofmann. *Extensional concepts in intensional type theory*. phdthesis, University of Edinburgh, January 1995.

33 Martin Hofmann and Thomas Streicher. The groupoid interpretation of type theory. In *Twenty-five years of constructive type theory (Venice, 1995)*, volume 36 of *Oxford Logic Guides*, pages 83–111. Oxford Univ. Press, 1998.

34 Simon Huber. Canonicity for Cubical Type Theory. *Journal of Automated Reasoning*, June 2018. doi:10.1007/s10817-018-9469-1.

35 Achim Jung and Jerzy Tiuryn. A new characterization of lambda definability. In Marc Bezem and Jan Friso Groote, editors, *Typed Lambda Calculi and Applications*, pages 245–257. Springer Berlin Heidelberg, 1993.

36 Ambrus Kaposi, András Kovács, and Thorsten Altenkirch. Constructing Quotient Inductive-inductive Types. *Proc. ACM Program. Lang.*, 3(POPL):2:1–2:24, January 2019. doi:10.1145/3290315.

37 Chris Kapulkin and Peter LeFanu Lumsdaine. The Simplicial Model of Univalent Foundations (after Voevodsky), June 2016. Preprint. arXiv:1211.2851.

38 Alexei Kopylov. *Type Theoretical Foundations for Data Structures, Classes and Objects*. phdthesis, Cornell University, 2004.

39 Peter LeFanu Lumsdaine and Michael Shulman. Semantics of higher inductive types, 2017. arXiv:1705.07088.

40 Jacob Lurie. *Higher Topos Theory*. Princeton University Press, 2009.

41 Per Martin-Löf. An Intuitionistic Theory of Types: Predicative Part. In H. E. Rose and J. C. Shepherdson, editors, *Logic Colloquium '73*, volume 80 of *Studies in Logic and the Foundations of Mathematics*, pages 73–118. Elsevier, 1975. doi:10.1016/S0049-237X(08)71945-1.

42 Per Martin-Löf. Constructive Mathematics and Computer Programming. In *6th International Congress for Logic, Methodology and Philosophy of Science*, pages 153–175, August 1979. Published by North Holland, Amsterdam. 1982.

43 Per Martin-Löf. *Intuitionistic type theory*, volume 1 of *Studies in Proof Theory*. Bibliopolis, 1984.

44 Conor McBride. *Dependently typed functional programs and their proofs.* phdthesis, University of Edinburgh, 1999.

45 Ieke Moerdijk and Erik Palmgren. Wellfounded trees in categories. *Annals of Pure and Applied Logic*, 104(1):189–218, 2000.

46 Bengt Nordström, Kent Peterson, and Jan M. Smith. *Programming in Martin-Löf's Type Theory*, volume 7 of *International Series of Monographs on Computer Science*. Oxford University Press, 1990.

47 Frank Pfenning. Intensionality, Extensionality, and Proof Irrelevance in Modal Type Theory. In *Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science*, LICS '01, pages 221–. IEEE Computer Society, 2001. URL: http://dl.acm.org/citation.cfm?id=871816.871845.

48 Michael Shulman. Univalence for inverse diagrams and homotopy canonicity. *Mathematical Structures in Computer Science*, 25(5):1203–1277, 2015. doi:10.1017/S0960129514000565.

49 Jonathan Sterling. Algebraic Type Theory and Universe Hierarchies, December 2018. arXiv:1902.08848.

50 Jonathan Sterling, Carlo Angiuli, and Daniel Gratzer. Cubical Syntax for Reflection-Free Extensional Equality, 2019. Extended version. arXiv:1904.08562.

51 Thomas Streicher. *Semantics of Type Theory: Correctness, Completeness, and Independence Results*. Birkhauser Boston Inc., 1991.

52 Thomas Streicher. Investigations Into Intensional Type Theory. Habilitationsschrift, Universität München, 1994.

53 Andrew Swan. Separating Path and Identity Types in Presheaf Models of Univalent Type Theory, 2018. arXiv:https://arxiv.org/abs/1808.00920.

54 Paul Taylor. *Practical Foundations of Mathematics*. Cambridge studies in advanced mathematics. Cambridge University Press, 1999.

55 The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. https://homotopytypetheory.org/book, 2013.

56 Vladimir Voevodsky. A simple type system with two identity types, February 2013. Talk at Andre Joyal's 70th birthday conference. (Slides available at https://www.math.ias.edu/vladimir/sites/math.ias.edu.vladimir/files/HTS_slides.pdf). URL: https://www.math.ias.edu/vladimir/sites/math.ias.edu.vladimir/files/HTS.pdf.

57 Vladimir Voevodsky. Mathematical theory of type theories and the initiality conjecture, April 2016. Research proposal to the Templeton Foundation for 2016-2019, project description. URL: http://www.math.ias.edu/Voevodsky/other/Voevodsky%20Templeton%20proposal.pdf.

A The rules of XTT

In the following sections, we summarize the rules of XTT; we systematically omit obvious premises to equational rules and all congruence rules for judgmental equality, because these can be mechanically obtained from the typing rules. We will write $\mathcal{J}_{=}$ schematically to refer to an equality judgment $\Psi \mid \Gamma \vdash A = B\ \mathit{type}_k$ or $\Psi \mid \Gamma \vdash M = N : A$.

$$\textsc{cube/emp}\ \frac{}{\cdot\ \mathit{cube}_+}\qquad \textsc{cube/snoc/dim}\ \frac{\Psi\ \mathit{cube}_+}{\Psi, i\ \mathit{cube}_+}\qquad \textsc{cube/snoc/constr}\ \frac{\Psi\ \mathit{cube}_+\quad \Psi \mid r, r'\ \mathit{dim}}{\Psi, r = r'\ \mathit{cube}_+}$$

$$\textsc{ctx/emp}\ \frac{}{\Psi \mid \cdot\ \mathit{ctx}}\qquad \textsc{ctx/snoc}\ \frac{\Psi \mid \Gamma\ \mathit{ctx}\quad \Psi \mid \Gamma \vdash A\ \mathit{type}_k}{\Psi \mid \Gamma, x : A\ \mathit{ctx}}$$

CONSTANT
$$\frac{}{\Psi \mid \varepsilon\ \mathit{dim}}$$

VARIABLE
$$\frac{i \in \Psi}{\Psi \mid i\ \mathit{dim}}$$

REFLEXIVITY
$$\frac{}{\Psi \mid r = r\ \mathit{dim}}$$

SYMM+TRANS
$$\frac{\Psi \mid r_0 = r_1\ \mathit{dim} \qquad \Psi \mid r_1 = r_2\ \mathit{dim}}{\Psi \mid r_2 = r_0\ \mathit{dim}}$$

HYP
$$\frac{\Psi \ni r = r'}{\Psi \mid r = r'\ \mathit{dim}}$$

VARIABLE
$$\frac{\Gamma \ni x : A}{\Psi \mid \Gamma \vdash x : A}$$

FALSE CONSTRAINT
$$\frac{\Psi \mid 0 = 1\ \mathit{dim}}{\Psi \mid \Gamma \vdash \mathcal{J}}$$

CONVERSION
$$\frac{\Psi \mid \Gamma \vdash A_0 = A_1\ \mathit{type}_k \qquad \Psi \mid \Gamma \vdash M : A_0}{\Psi \mid \Gamma \vdash M : A_1}$$

BOUNDARY SEPARATION
$$\frac{\Psi \mid r\ \mathit{dim} \qquad \overrightarrow{\Psi, r = \varepsilon \mid \Gamma \vdash \mathcal{J}_=}}{\Psi \mid \Gamma \vdash \mathcal{J}_=}$$

COERCION
$$\frac{\Psi \mid r, r'\ \mathit{dim} \qquad \Psi, i \mid \Gamma \vdash A\ \mathit{type}_k \qquad \Psi \mid \Gamma \vdash M : A\langle r/i\rangle}{\Psi \mid \Gamma \vdash [i.A]\downarrow^r_{r'} M : A\langle r'/i\rangle \qquad \Psi \mid \Gamma \vdash [i.A]\downarrow^r_r M = M : A\langle r/i\rangle}$$

COERCION REGULARITY
$$\frac{\Psi, j, j' \mid \Gamma \vdash A\langle j/i\rangle = A\langle j'/i\rangle\ \mathit{type}_k}{\Psi \mid \Gamma \vdash [i.A]\downarrow^r_{r'} M = M : A\langle r'/i\rangle}$$

COMPOSITION
$$\frac{\Psi \mid r, r', s\ \mathit{dim} \qquad \Psi \mid \Gamma \vdash M : A \qquad \overrightarrow{\Psi, j, s = \varepsilon \mid \Gamma \vdash N_\varepsilon : A\ [j = r \hookrightarrow M]}}{\begin{array}{c}\Psi \mid \Gamma \vdash A\downarrow^r_{r'} M\ [s\ \textsf{with}\ \overrightarrow{\varepsilon \hookrightarrow j.N_\varepsilon}] : A \\ \Psi \mid \Gamma \vdash A\downarrow^r_{r} M\ [s\ \textsf{with}\ \overrightarrow{\varepsilon \hookrightarrow j.N_\varepsilon}] = M : A \\ \Psi \mid \Gamma \vdash A\downarrow^r_{r'} M\ [\varepsilon\ \textsf{with}\ \overrightarrow{\varepsilon' \hookrightarrow j.N_{\varepsilon'}}] = N_\varepsilon\langle r'/j\rangle : A\end{array}}$$

LIFT FORMATION
$$\frac{\Psi \mid \Gamma \vdash A\ \mathit{type}_k \qquad k \leq l}{\begin{array}{l}\Psi \mid \Gamma \vdash \Uparrow^l_k A\ \mathit{type}_l \\ \Psi \mid \Gamma \vdash \Uparrow^k_k A = A\ \mathit{type}_k \\ \Psi \mid \Gamma \vdash \Uparrow^m_l \Uparrow^l_k A = \Uparrow^m_k A\ \mathit{type}_k\end{array}}$$

LIFT ELEMENT
$$\frac{\Psi \mid \Gamma \vdash M : A}{\Psi \mid \Gamma \vdash M : \Uparrow^l_k A}$$

LIFT HYPOTHESIS
$$\frac{\Psi \mid \Gamma, x : A \vdash \mathcal{J}}{\Psi \mid \Gamma, x : \Uparrow^l_k A \vdash \mathcal{J}}$$

LIFT COERCION
$$\frac{}{\Psi \mid \Gamma \vdash [i.\Uparrow^l_k A]\downarrow^r_{r'} M = [i.A]\downarrow^r_{r'} M : \Uparrow^l_k A\langle r'/i\rangle}$$

PAIR FORMATION, LIFTING
$$\frac{\Psi \mid \Gamma \vdash A\ \mathit{type}_k \qquad \Psi \mid \Gamma, x : A \vdash B\ \mathit{type}_k}{\begin{array}{l}\Psi \mid \Gamma \vdash (x : A) \times B\ \mathit{type}_k \\ \Psi \mid \Gamma \vdash \Uparrow^l_k (x : A) \times B = (x : \Uparrow^l_k A) \times \Uparrow^l_k B\ \mathit{type}_l\end{array}}$$

PAIR INTRODUCTION
$$\frac{\Psi \mid \Gamma \vdash A\ \mathit{type}_k \qquad \Psi \mid \Gamma, x : A \vdash B\ \mathit{type}_k \qquad \Psi \mid \Gamma \vdash M : A \qquad \Psi \mid \Gamma \vdash N : B[M/x]}{\Psi \mid \Gamma \vdash \langle M, N\rangle : (x : A) \times B}$$

PAIR ELIMINATION

$$\frac{\Psi \mid \Gamma \vdash A\ type_k \quad \Psi \mid \Gamma, x : A \vdash B\ type_k \quad \Psi \mid \Gamma \vdash M : (x : A) \times B}{\begin{array}{l}\Psi \mid \Gamma \vdash \mathsf{fst}_{x:A.B}(M) : A \\ \Psi \mid \Gamma \vdash \mathsf{snd}_{x:A.B}(M) : B[\mathsf{fst}(M)/x] \\ \Psi \mid \Gamma \vdash \mathsf{fst}_{x:\Uparrow^l_k A.\Uparrow^l_k B}(M) = \mathsf{fst}_{x:A.B}(M) : A \\ \Psi \mid \Gamma \vdash \mathsf{snd}_{x:\Uparrow^l_k A.\Uparrow^l_k B}(M) = \mathsf{snd}_{x:A.B}(M) : B[\mathsf{fst}(M)/x]\end{array}}$$

PAIR COMPUTATION

$$\frac{\Psi \mid \Gamma \vdash H \triangleq [i.B[[i.A] \downarrow^r_i \mathsf{fst}(M)/x]] \downarrow^r_{r'} \mathsf{snd}(M)}{\begin{array}{l}\Psi \mid \Gamma \vdash \mathsf{fst}(\langle M, N\rangle) = M : A \\ \Psi \mid \Gamma \vdash \mathsf{snd}(\langle M, N\rangle) = N : B[M/x] \\ \Psi \mid \Gamma \vdash \mathsf{fst}([i.(x : A) \times B] \downarrow^r_{r'} M) = [i.A] \downarrow^r_{r'} \mathsf{fst}(M) : A\langle r'/i\rangle \\ \Psi \mid \Gamma \vdash \mathsf{snd}([i.(x : A) \times B] \downarrow^r_{r'} M) = H : B\langle r'/i\rangle[[i.A] \downarrow^r_{r'} \mathsf{fst}(M)/x]\end{array}}$$

PAIR UNICITY

$$\frac{}{\Psi \mid \Gamma \vdash M = \langle \mathsf{fst}(M), \mathsf{snd}(M)\rangle : (x : A) \times B}$$

FUNCTION FORMATION, LIFTING

$$\frac{\begin{array}{c}\Psi \mid \Gamma \vdash A\ type_k \\ \Psi \mid \Gamma, x : A \vdash B\ type_k\end{array}}{\begin{array}{l}\Psi \mid \Gamma \vdash (x : A) \to B\ type_k \\ \Psi \mid \Gamma \vdash \Uparrow^l_k (x : A) \to B = (x : \Uparrow^l_k A) \to \Uparrow^l_k B\ type_l\end{array}}$$

FUNCTION INTRODUCTION

$$\frac{\begin{array}{l}\Psi \mid \Gamma \vdash A\ type_k \\ \Psi \mid \Gamma, x : A \vdash B\ type_k \\ \Psi \mid \Gamma, x : A \vdash M : B\end{array}}{\Psi \mid \Gamma \vdash \lambda x.M : (x : A) \to B}$$

FUNCTION ELIMINATION

$$\frac{\begin{array}{c}\Psi \mid \Gamma \vdash A\ type_k \quad \Psi \mid \Gamma, x : A \vdash B\ type_k \\ \Psi \mid \Gamma \vdash M : (x : A) \to B \quad \Psi \mid \Gamma \vdash N : A\end{array}}{\begin{array}{l}\Psi \mid \Gamma \vdash \mathsf{app}_{x:A.B}(M, N) : B[N/x] \\ \Psi \mid \Gamma \vdash \mathsf{app}_{x:\Uparrow^l_k A.\Uparrow^l_k B}(M, N) = \mathsf{app}_{x:A.B}(M, N) : x[N/B]\end{array}}$$

FUNCTION COMPUTATION

$$\frac{\Psi, i \mid \Gamma \vdash \widetilde{N}[i] \triangleq [i.A] \downarrow^{r'}_i N}{\begin{array}{l}\Psi \mid \Gamma \vdash (\lambda x.M)(N) = M[N/x] : B[N/x] \\ \Psi \mid \Gamma \vdash ([i.(x : A) \to B] \downarrow^r_{r'} M)(N) = [i.B[\widetilde{N}[i]/x]] \downarrow^r_{r'} M(\widetilde{N}[r]) : C\end{array}}$$

FUNCTION UNICITY

$$\frac{}{\Psi \mid \Gamma \vdash M = \lambda x.M(x) : (x : A) \to B}$$

EQUALITY FORMATION, LIFTING

$$\frac{\Psi, i \mid \Gamma \vdash A\ type_k \quad \overrightarrow{\Psi, i, i = \varepsilon \mid \Gamma \vdash N_\varepsilon : A}}{\begin{array}{l}\Psi \mid \Gamma \vdash \mathsf{Eq}_{i.A}(N_0, N_1)\ type_k \\ \Psi \mid \Gamma \vdash \Uparrow^l_k \mathsf{Eq}_{i.A}(N_0, N_1) = \mathsf{Eq}_{i.\Uparrow^l_k A}(N_0, N_1)\ type_l\end{array}}$$

EQUALITY INTRODUCTION

$$\frac{\Psi, i \mid \Gamma \vdash M : A\ [\overrightarrow{i = \varepsilon \hookrightarrow N_\varepsilon}]}{\Psi \mid \Gamma \vdash \lambda i.M : \mathsf{Eq}_{i.A}(N_0, N_1)}$$

EQUALITY ELIMINATION

$$\frac{\Psi \mid r\ dim \quad \Psi, i \mid \Gamma \vdash A\ type_k \quad \overrightarrow{\Psi, i, i = \varepsilon \mid \Gamma \vdash N_\varepsilon : A} \quad \Psi \mid \Gamma \vdash M : \mathsf{Eq}_{i.A}(N_0, N_1)}{\begin{array}{l}\Psi \mid \Gamma \vdash \mathsf{app}_{i.A}(M, r) : A\langle r/i\rangle \\ \Psi \mid \Gamma \vdash \mathsf{app}_{i.\Uparrow^l_k A}(M, r) = \mathsf{app}_{i.A}(M, r) : A\langle r/i\rangle \\ \Psi \mid \Gamma \vdash \mathsf{app}_{i.A}(M, \varepsilon) = N_\varepsilon : A\langle \varepsilon/i\rangle\end{array}}$$

EQUALITY COMPUTATION

$$\frac{}{\begin{array}{l}\Psi \mid \Gamma \vdash (\lambda i.M)(r) = M\langle r/i\rangle : A\langle r/i\rangle \\ \Psi \mid \Gamma \vdash ([j.\mathsf{Eq}_{i.A}(N_0, N_1)] \downarrow^{r}_{r'} P)(s) = [j.A\langle s/i\rangle] \downarrow^{r}_{r'} P(s)\ [s \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.N_\varepsilon}] : A\langle r', s/j, i\rangle\end{array}}$$

EQUALITY UNICITY

$$\frac{}{\Psi \mid \Gamma \vdash M = \lambda i.M(i) : \mathsf{Eq}_{i.A}(N_0, N_1)}$$

BOOLEAN FORMATION, LIFTING, INTRODUCTION

$$\frac{}{\begin{array}{l}\Psi \mid \Gamma \vdash \mathsf{bool}\ \mathit{type}_k \\ \Psi \mid \Gamma \vdash \Uparrow^{l}_{k}\mathsf{bool} = \mathsf{bool}\ \mathit{type}_l \\ \Psi \mid \Gamma \vdash \mathsf{true} : \mathsf{bool} \\ \Psi \mid \Gamma \vdash \mathsf{false} : \mathsf{bool}\end{array}}$$

BOOLEAN ELIMINATION

$$\frac{\begin{array}{c}\Psi \mid \Gamma, x : \mathsf{bool} \vdash C\ \mathit{type}_k \\ \Psi \mid \Gamma \vdash M : \mathsf{bool} \\ \Psi \mid \Gamma \vdash N_0 : C[\mathsf{true}/x] \\ \Psi \mid \Gamma \vdash N_1 : C[\mathsf{false}/x]\end{array}}{\Psi \mid \Gamma \vdash \mathsf{if}_{x.C}(M; N_0, N_1) : C[M/x]}$$

BOOLEAN ELIMINATION LIFTING

$$\frac{}{\Psi \mid \Gamma \vdash \mathsf{if}_{x.\Uparrow^{l}_{k} C}(M; N_0, N_1) = \mathsf{if}_{x.C}(M; N_0, N_1) : \Uparrow^{l}_{k} C[M/x]}$$

BOOLEAN COMPUTATION

$$\frac{}{\begin{array}{l}\Psi \mid \Gamma \vdash \mathsf{if}_{x.C}(\mathsf{true}; N_0, N_1) = N_0 : C[\mathsf{true}/x] \\ \Psi \mid \Gamma \vdash \mathsf{if}_{x.C}(\mathsf{false}; N_0, N_1) = N_1 : C[\mathsf{false}/x]\end{array}}$$

UNIVERSE FORMATION, LIFTING

$$\frac{k < l}{\begin{array}{l}\Psi \mid \Gamma \vdash \mathcal{U}_k\ \mathit{type}_l \\ \Psi \mid \Gamma \vdash \Uparrow^{m}_{l}\mathcal{U}_k = \mathcal{U}_k\ \mathit{type}_m\end{array}}$$

UNIVERSE ELEMENTS

$$\frac{\Psi \mid \Gamma \vdash A\ \mathit{type}_k}{\Psi \mid \Gamma \vdash A : \mathcal{U}_k}$$

UNIVERSE EQUALITY

$$\frac{\Psi \mid \Gamma \vdash A_0 = A_1\ \mathit{type}_k}{\Psi \mid \Gamma \vdash A_0 = A_1 : \mathcal{U}_k}$$

TYPE-CASE

$$\frac{\begin{array}{l}\Psi \mid \Gamma \vdash C\ \mathit{type}_l \\ \Psi \mid \Gamma, x : \mathcal{U}_k, y : x \to \mathcal{U}_k \vdash M_\Pi : C \\ \Psi \mid \Gamma, x : \mathcal{U}_k, y : x \to \mathcal{U}_k \vdash M_\Sigma : C \\ \Psi \mid \Gamma, x_0 : \mathcal{U}_k, x_1 : \mathcal{U}_k, x^{=} : \mathsf{Eq}_{i.\mathcal{U}_k}(x_0, x_1), y_0 : x_0, y_1 : x_1 \vdash M_{\mathsf{Eq}} : C \\ \Psi \mid \Gamma \vdash M_{\mathsf{bool}} : C \\ \Psi \mid \Gamma \vdash M_{\mathcal{U}} : C\end{array}}{\Psi \mid \Gamma \vdash \mathsf{tycase}\ X\ [\Pi_x y \mapsto M_\Pi \mid \Sigma_x y \mapsto M_\Sigma \mid \mathsf{Eq}_{x_0, x_1, x^{=}}(y_0, y_1) \mapsto M_{\mathsf{Eq}} \mid \mathsf{bool} \mapsto M_{\mathsf{bool}} \mid \mathcal{U} \mapsto M_{\mathcal{U}}] : C}$$

TYPE-CASE COMPUTATION

$$\frac{\Psi \mid \Gamma \vdash H_{\mathbf{Eq}} \triangleq M[A\langle 0/i\rangle, A\langle 1/i\rangle, \lambda i.A, N_0, N_1/x_0, x_1, x^{=}, y_0, y_1]}{\begin{array}{l}\Psi \mid \Gamma \vdash \mathsf{tycase}\ ((z : A) \to B)\ [\Pi_x y \mapsto M \mid \ldots] = M[A, \lambda z.B/x, y] : C \\ \Psi \mid \Gamma \vdash \mathsf{tycase}\ ((z : A) \times B)\ [\ldots \mid \Sigma_x y \mapsto M \mid \ldots] = M[A, \lambda z.B/x, y] : C \\ \Psi \mid \Gamma \vdash \mathsf{tycase}\ \mathsf{bool}\ [\ldots \mid \mathsf{bool} \mapsto M \mid \ldots] = M : C \\ \Psi \mid \Gamma \vdash \mathsf{tycase}\ \mathcal{U}_{k'}\ [\ldots \mid \mathcal{U} \mapsto M] = M : C \\ \Psi \mid \Gamma \vdash \mathsf{tycase}\ (\mathsf{Eq}_{i.A}(N_0, N_1))\ [\ldots \mid \mathsf{Eq}_{x_0, x_1, x^{=}}(y_0, y_1) \mapsto M \mid \ldots] = H_{\mathbf{Eq}} : C\end{array}}$$

TYPE BOUNDARY

$$\frac{\Psi \mid \Gamma \vdash M : A \qquad \overrightarrow{\Psi, \xi \mid \Gamma \vdash M = N : A}}{\Psi \mid \Gamma \vdash M : A\ [\overrightarrow{\xi \hookrightarrow N}]}$$

TERM BOUNDARY

$$\frac{\Psi \mid \Gamma \vdash A\ \mathit{type}_k \qquad \overrightarrow{\Psi, \xi \mid \Gamma \vdash A = B\ \mathit{type}_k}}{\Psi \mid \Gamma \vdash A\ \mathit{type}_k\ [\overrightarrow{\xi \hookrightarrow B}]}$$

A.1 Derivable Rules

Numerous additional rules about compositions are *derivable* by exploiting boundary separation. In previous presentations of cubical type theory (which did not enjoy the unicity of equality proofs), it was necessary to include β-rules for compositions explicitly.

COMPOSITION REGULARITY

$$\frac{\overrightarrow{\Psi, j_0, j_1, i = \varepsilon \mid \Gamma \vdash N_\varepsilon\langle j_0/j\rangle = N_\varepsilon\langle j_1/j\rangle : A}}{\Psi \mid \Gamma \vdash A \downarrow^r_{r'} M\ [i \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.N_\varepsilon}] = M : A}$$

HETEROGENEOUS COMPOSITION

$$\frac{\Psi \mid r, r', s\ \mathit{dim} \quad \Psi \mid \Gamma \vdash M : A\langle r/j\rangle \quad \overrightarrow{\Psi, j, s = \varepsilon \mid \Gamma \vdash N_\varepsilon : A\ [j = r \hookrightarrow M]}}{\begin{array}{l}\Psi \mid \Gamma \vdash [j.A] \downarrow^r_{r'} M\ [s \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.N_\varepsilon}] : A\langle r'/j\rangle \\ \Psi \mid \Gamma \vdash [j.A] \downarrow^r_{r} M\ [s \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.N_\varepsilon}] = M : A\langle r/j\rangle \\ \Psi \mid \Gamma \vdash [j.A] \downarrow^r_{r'} M\ [\varepsilon \text{ with } \overrightarrow{\varepsilon' \hookrightarrow j.N_{\varepsilon'}}] = N_\varepsilon\langle r'/j\rangle : A\langle r'/j\rangle\end{array}}$$

LIFT COMPOSITION

$$\frac{}{\Psi \mid \Gamma \vdash {\Uparrow^l_k} A \downarrow^r_{r'} M\ [i \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.N_\varepsilon}] = A \downarrow^r_{r'} M\ [i \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.N_\varepsilon}] : {\Uparrow^l_k} A}$$

LIFT TYPE COMPOSITION

$$\frac{}{\Psi \mid \Gamma \vdash \mathcal{U}_l \downarrow^r_{r'} {\Uparrow^l_k} A\ [i \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.{\Uparrow^l_k} B_\varepsilon}] = {\Uparrow^l_k} \mathcal{U}_k \downarrow^r_{r'} A\ [i \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.B_\varepsilon}]\ \mathit{type}_k}$$

PAIR COMPOSITION COMPUTATION (1)

$$\frac{\Psi \mid \Gamma \vdash H \triangleq A \downarrow^r_{r'} \mathsf{fst}(M)\ [i \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.\mathsf{fst}(N_\varepsilon)}]}{\Psi \mid \Gamma \vdash \mathsf{fst}((x : A) \times B \downarrow^r_{r'} M\ [i \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.N_\varepsilon}]) = H : A}$$

PAIR COMPOSITION COMPUTATION (2)

$$\frac{\begin{array}{c}\Psi, k \mid \Gamma \vdash \widetilde{M_1}[k] \triangleq A \downarrow^r_k \mathsf{fst}(M)\ [i \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.\mathsf{fst}(N_\varepsilon)}] \\ \Psi \mid \Gamma \vdash H \triangleq [k.B[\widetilde{M_1}[k]/x]] \downarrow^r_{r'} \mathsf{snd}(M)\ [i \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.\mathsf{snd}(N_\varepsilon)}]\end{array}}{\Psi \mid \Gamma \vdash \mathsf{snd}((x : A) \times B \downarrow^r_{r'} M\ [i \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.N_\varepsilon}]) = H : B[\widetilde{M_1}[r']/x]}$$

PAIR TYPE COMPOSITION

$$\frac{\begin{array}{l}\Psi, k \mid \Gamma \vdash \widetilde{A}[k] \triangleq \mathcal{U}_k \downarrow^r_k A\ [i \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.A_\varepsilon}] \\ \Psi, j \mid \Gamma, x : \widetilde{A}[r'] \vdash \widetilde{x}[j] \triangleq [k.\widetilde{A}[k]] \downarrow^{r'}_j x \\ \Psi \mid \Gamma, x : \widetilde{A}[r'] \vdash \widetilde{B} \triangleq \mathcal{U}_k \downarrow^r_{r'} B[\widetilde{x}[r]/x]\ [i \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.B_\varepsilon[\widetilde{x}[j]/x]}]\end{array}}{\Psi \mid \Gamma \vdash \mathcal{U}_k \downarrow^r_{r'} ((x : A) \times B)\ [i \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.(x : A_\varepsilon) \times B_\varepsilon}] = (x : \widetilde{A}[r']) \times \widetilde{B}\ \mathit{type}_l}$$

FUNCTION COMPOSITION COMPUTATION

$$\frac{\Psi \mid \Gamma \vdash H \triangleq B[N/x] \downarrow^r_{r'} M(N)\ [i \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.M_\varepsilon(N)}]}{\Psi \mid \Gamma \vdash ((x : A) \to B \downarrow^r_{r'} M\ [i \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.M_\varepsilon}])(N) = H : (x : A) \to B}$$

FUNCTION TYPE COMPOSITION

$$\frac{\begin{array}{c}\Psi, k \mid \Gamma \vdash \widetilde{A}[k] \triangleq \mathcal{U}_k \downarrow_k^r A\ [i \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.A_\varepsilon}] \\ \Psi, j \mid \Gamma, x : \widetilde{A}[r'] \vdash \widetilde{x}[j] \triangleq [k.\widetilde{A}[k]] \downarrow_j^{r'} x \\ \Psi \mid \Gamma, x : \widetilde{A}[r'] \vdash \widetilde{B} \triangleq \mathcal{U}_k \downarrow_{r'}^r B[\widetilde{x}[r]/x]\ [i \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.B_\varepsilon[\widetilde{x}[j]/x]}]\end{array}}{\Psi \mid \Gamma \vdash \mathcal{U}_k \downarrow_{r'}^r ((x : A) \to B)\ [i \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.(x : A_\varepsilon) \to B_\varepsilon}] = (x : \widetilde{A}[r']) \to \widetilde{B}\ \mathit{type}_l}$$

EQUALITY COMPOSITION COMPUTATION

$$\frac{\Psi \mid \Gamma \vdash H \triangleq A\langle s/i\rangle \downarrow_{r'}^r P(s)\ [k \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.Q_\varepsilon(s)}]}{\Psi \mid \Gamma \vdash (\mathsf{Eq}_{i.A}(N_0, N_1) \downarrow_{r'}^r P\ [k \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.Q_\varepsilon}])(s) = H : A\langle s/i\rangle}$$

EQUALITY TYPE COMPOSITION

$$\frac{\begin{array}{c}\Psi, j, i \mid \Gamma \vdash \widetilde{A}[j, i] \triangleq \mathcal{U}_k \downarrow_j^r A\ [k \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.A_\varepsilon}] \\ \Psi \mid \Gamma \vdash \widetilde{M} \triangleq [j.\widetilde{A}[j, r]] \downarrow_{r'}^r M\ [k \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.M_\varepsilon}] \\ \Psi \mid \Gamma \vdash \widetilde{N} \triangleq [j.\widetilde{A}[j, r']] \downarrow_{r'}^r N\ [k \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.N_\varepsilon}]\end{array}}{\Psi \mid \Gamma \vdash \mathcal{U}_k \downarrow_{r'}^r \mathsf{Eq}_{i.A}(M, N)\ [k \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.\mathsf{Eq}_{i.A_\varepsilon}(M_\varepsilon, N_\varepsilon)}] = \mathsf{Eq}_{i.\widetilde{A}[r', i]}(\widetilde{M}, \widetilde{N})\ \mathit{type}_l}$$

BASE TYPE COMPOSITION

$$\frac{(\mathfrak{b} \in \{\mathsf{bool}, \mathcal{U}_{k'}\})}{\Psi \mid \Gamma \vdash \mathcal{U}_k \downarrow_{r'}^r \mathfrak{b}\ [i \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.\mathfrak{b}}] = \mathfrak{b}\ \mathit{type}_l}$$

Guarded Recursion in Agda via Sized Types

Niccolò Veltri

Department of Computer Science, IT University of Copenhagen, Denmark
nive@itu.dk

Niels van der Weide

Institute for Computation and Information Sciences, Radboud University
Nijmegen, The Netherlands
nweide@cs.ru.nl

Abstract

In type theory, programming and reasoning with possibly non-terminating programs and potentially infinite objects is achieved using coinductive types. Recursively defined programs of these types need to be productive to guarantee the consistency of the type system. Proof assistants such as Agda and Coq traditionally employ strict syntactic productivity checks, which often make programming with coinductive types convoluted. One way to overcome this issue is by encoding productivity at the level of types so that the type system forbids the implementation of non-productive corecursive programs. In this paper we compare two different approaches to type-based productivity: guarded recursion and sized types. More specifically, we show how to simulate guarded recursion in Agda using sized types. We formalize the syntax of a simple type theory for guarded recursion, which is a variant of Atkey and McBride's calculus for productive coprogramming. Then we give a denotational semantics using presheaves over the preorder of sizes. Sized types are fundamentally used to interpret the characteristic features of guarded recursion, notably the fixpoint combinator.

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

Keywords and phrases guarded recursion, type theory, semantics, coinduction, sized types

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.32

Funding *Niccolò Veltri*: Veltri was supported by a research grant (13156) from VILLUM FONDEN.

Acknowledgements We are thankful to Andreas Abel, Guillaume Allais, Herman Geuvers, Rasmus Ejlers Møgelberg and Andrea Vezzosi for discussions and valuable hints. We thank the anonymous referees for their useful comments.

1 Introduction

Dependent type theory is an expressive functional programming language that underlies the deductive system of proof assistants such as Agda [22] and Coq [10]. It is a total language, meaning that every program definable inside type theory is necessarily terminating. This is an important requirement that ensures the consistency of the type system.

Possibly non-terminating computations and infinite structures, such as non-wellfounded trees, can be represented in type theory by extending the type system with coinductive types. Recursively defined elements of these types need to be productive in the sense that every finite part of the output can be computed in a finite number of steps [15]. In Agda's encoding of coinductive types using "musical notation" [16], productivity is enforced via a strict obligation: in the definition of a corecursive function, recursive calls must appear directly under the application of a constructor. A similar syntactic check is used in Coq. This restriction typically makes programming with coinductive types cumbersome, which spawned the search for alternative techniques to ensure the well-definedness of corecursive definitions.

© Niccolò Veltri and Niels van der Weide;
licensed under Creative Commons License CC-BY

4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019).
Editor: Herman Geuvers; Article No. 32; pp. 32:1–32:19
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

We focus on two of these techniques in which productivity is encoded at the level of types: sized types and guarded recursion. A sized type is a type annotated with the number of unfoldings that elements of this type can undergo [17]. Sized types have been implemented in Agda and can be used in combination with coinductive records to specify coinductive types [4, 6], as exemplified by Abel and Chapman's work [3]. There also are other approaches to sized types. One of those is developed by Sacchini [23], which allows less terms to be typed compared to Agda's approach.

Guarded recursion [21] is a different approach where the type system is enhanced with a modality, called "later" and written $\triangleright$, encoding time delay in types. The later modality comes with a general fixpoint combinator for programming with productive recursive functions and allows the specification of guarded recursive types. These types can be used in combination with clock quantification to define coinductive types [8, 9, 12]. Currently there is no implementation of a calculus for guarded recursion.

Sized types and guarded recursion are different solutions to the same problem, but a thorough study of the relation between these two disciplines is still missing. In this paper we take a first step in this direction by showing how guarded recursion can be simulated in Agda using sized types.

Utilizing techniques for representing "type theory in type theory" [7, 13], we present an Agda formalization of the syntax of a simple type theory for guarded recursion. This object language, which we call **GTT**, is a variant of Atkey and McBride's type system for productive coprogramming [8] in which the clock context can contain at most one clock. The types of **GTT** include the aforementioned $\triangleright$ modality and a $\square$ modality, a nameless analogue of Atkey and McBride's universal clock quantification. Clouston *et al.* [14] studied a guarded variant of lambda calculus extended with a $\square$ operation, which they call "constant". Our object calculus differs from theirs in that our judgments are indexed by a clock context, which can be empty or containing exactly one clock. The design of **GTT** has the benefit of allowing a more appealing introduction rule for $\square$ than Clouston *et al.*'s. **GTT** also differs from Clouston *et al.*'s calculus in the class of definable guarded recursive types. We follow Atkey and McBride's approach and restrict the least fixpoint operator μ to act on strictly positive functors, while μ type former in Clouston *et al.*solely operates on functors in which all variables are guarded by an occurrence of the $\triangleright$ modality.

Afterwards we develop a categorical semantics for **GTT** using sized types. More precisely, we define a presheaf model where the type formers of simply typed lambda calculus are evaluated using the standard Kripke semantics. Typically, the semantics of a type theory for guarded recursion is given in the topos of trees or variations thereof [11, 19, 20]. Here, to clarify the connection between guarded recursion and sized types, we take the preorder of sizes as the indexing category of our presheaves. This means that types and contexts of **GTT** are interpreted as antitone sized types. The well-order relation on sizes is fundamental for constructing a terminating definition of the semantic fixpoint combinator.

The decision to restrict the clock context of **GTT** to contain at most one clock variable was dictated by our choice to use Agda with sized types as the metatheory. In Agda, it is possible to encode semantic clock contexts containing multiple clocks as lists of sizes or as functions from a finite interval of natural numbers into sizes. However, Agda's support for sized types is tailored to types depending on exactly one size, or on a finite but precise number of sizes, which makes it cumbersome to work with types depending on clock contexts containing an indefinite number of clocks. More technically, as it was privately communicated to us by Agda implementors, Agda's type inference only works properly for first-order size constraints. We believe that if future Agda releases add the capability of handling multiple

sizes, it would be possible to extend the semantics of **GTT** to include the full Atkey and McBride's type theory. We also believe that extending our formalization to the multiclock case, while technically challenging, would not be conceptually harder.

GTT is stricly less expressive than Atkey and McBride's calculus, since we are unable to implement nested coinductive types. But the one clock variant of the calculus still allows the construction of a large class of coinductive types and it is the subject of active research [14].

To summarize, the contributions of this paper are twofold:

1. We formalize syntax and semantics of a type theory for guarded recursion in Agda. This is the first such denotational model developed using an extension of Martin-Löf intensional type theory as the metalanguage, in contrast with the previous set-theoretic models of guarded recursion.
2. The interpretation of the characteristic features of guarded recursion crucially requires the employment of sized types. This shows that guarded recursion can be reduced to sized types and constitutes a stepping stone towards a clear understanding of the connections between different approaches to productive coprogramming.

This paper only include the essential parts of our Agda formalization. The full code is available at `https://github.com/niccoloveltri/agda-gtt`. The formalization uses Agda 2.5.4.2 and Agda standard library 0.16. The paper is extracted from a literate Agda file, which implies that all the displayed code passed Agda's type and termination checker.

The material is organized in the following way. In Section 2, we discuss the metatheory and we give an overview of sized types. In Section 3, we introduce the syntax of **GTT**, our object type theory for guarded recursion. Subsequently, we give a categorical semantics. In Section 4, we show how to implement presheaves over sizes and how to model the fragment of **GTT** corresponding to simply typed lambda calculus. In Section 5, we discuss the semantics of the guarded recursive and coinductive features. In Section 6, we prove the object language sound w.r.t. the categorical model, which in turn entails the consistency of the syntax. Finally, in Section 7, we draw conclusions and suggest future directions of work.

2 The Host Type Theory

We work in Martin-Löf type theory extended with functional extensionality, uniqueness of identity proofs (UIP), and sized types. Practically, we work in Agda, which supports sized types and where UIP holds by default. In this section, we give a brief overview of these principles and we introduce the basic Agda notation that we employ in our formalization.

We write $=$ for judgmental equality and $\equiv$ for propositional equality. Implicit arguments of functions are delimited by curly brackets. We write $\forall \{\Delta\}$ for an implicit argument Δ whose type can be inferred by Agda. We write Set, Set_1 and Set_2 for the first three universes of types. We write $\bot$ for the empty type.

We make extensive use of record types. These are like iterated Σ-types, in which each component, also called field, has been given a name. We open each record we introduce, which allows us to access a field by function application. For example, given a record type R containing a field f of type A, we have f R : A. We use Agda's copatterns for defining elements of a record type. If a record type R contains fields $\mathsf{f}_1 : A_1$ and $\mathsf{f}_2 : A_2$, we construct a term r : R by specifying its components f_1 r : A_1 and f_2 r : A_2.

The principle of functional extensionality states that every two functions f and g in the same function space are equal whenever $f\ x$ and $g\ x$ are equal for all inputs x. This principle is not provable in Agda, so we need to postulate it. UIP states that all proofs of an identity are propositionally equal. Agda natively supports this principle, which is therefore derivable.

Agda also natively supports sized types [2, 6]. Intuitively, a sized type is a type annotated with an abstract ordinal indicating the number of possible unfoldings that can be performed of elements of the type. These abstract ordinals, called sizes, assist the termination checker in assessing the well-definedness of corecursive definitions.

In Agda, there is a type Size of sizes and a type Size< i of sizes strictly smaller than i. Every size j : Size< i is coerced to j : Size. The order relation on sizes is transitive, which means that whenever j : Size< i and k : Size< j, then k : Size< i. The order relation is also well-founded, which is used to define productive corecursive functions. There is a successor operation ↑ on sizes and a size ∞ such that i : Size< ∞ for all i. Lastly, we define a sized type to be a type indexed by Size.

3 The Object Type Theory

The object language we consider is simply typed lambda calculus extended with additional features for programming with guarded recursive and coinductive types. We call this language **GTT**. It is a variant of Atkey and McBride's type system for productive coprogramming [8]. In Atkey and McBride's calculus, all judgments are indexed by a clock context, which may contain several different clocks. They extend simply typed lambda calculus with two additional type formers: a modality $\triangleright$ for encoding time delay into types and universal quantification over clock variables $\forall$, which is used in combination with $\triangleright$ to specify coinductive types. In **GTT**, judgments are also indexed by a clock context, but in our case the latter can contain at most one clock variable κ. The type system of **GTT** also includes a $\triangleright$ modality, plus a box modality corresponding to Atkey and McBride's quantification over the clock variable κ.

In this section we introduce the syntax of **GTT** as in our Agda formalization. We give a more standard presentation of the calculus in Appendix A.

GTT is a type theory with explicit substitutions [1]. It comprises well-formed types and contexts, well-typed terms and substitutions, definitional equality of terms and of substitutions. All of them depend on a clock context. In **GTT**, the clock context can either be empty or contain a single clock κ.

```
data ClockCtx : Set where
  ∅ : ClockCtx
  κ : ClockCtx
```

We refer to types and contexts in the empty clock context as ∅-types and ∅-contexts respectively. Similarly, κ-types and κ-contexts are types and contexts depending on κ.

3.1 Types

The well-formed types of **GTT** include the unit type, products, coproducts, and function spaces. Notice that **1** is a ∅-type.

```
data Ty : ClockCtx → Set where
  1 : Ty ∅
  _⊠_ : ∀ {Δ} → Ty Δ → Ty Δ → Ty Δ
  _⊞_ : ∀ {Δ} → Ty Δ → Ty Δ → Ty Δ
  _⟶_ : ∀ {Δ} → Ty Δ → Ty Δ → Ty Δ
```

We include a modality $\triangleright$ as an operation on κ-types similar to the one in Atkey and McBride's system. There also is a nameless analogue of clock quantification, which we call "box" and denote by □ following [14]. The box modality takes a κ-type and returns a ∅-type.

The well-formed types of **GTT** include a weakening type former ⇡, which maps ∅-types to κ-types.

```
▷ : Ty κ → Ty κ
□ : Ty κ → Ty ∅
⇡ : Ty ∅ → Ty κ
```

Guarded recursive types are defined using a least fixpoint type former μ.

```
μ : ∀ {Δ} → Code Δ → Ty Δ
```

For μ to be well-defined, one typically limits its applicability to strictly positive functors. We instead consider a grammar Code Δ for functors, which has codes for constant functors, the identity, products, coproducts, and the later modality. Since there is a code for constant functors, the type family Code needs to be defined simultaneously with Ty.

```
data Code : ClockCtx → Set where
  ℂ : ∀ {Δ} → Ty Δ → Code Δ
  I : ∀ {Δ} → Code Δ
  _⊠_ : ∀ {Δ} → Code Δ → Code Δ → Code Δ
  _⊞_ : ∀ {Δ} → Code Δ → Code Δ → Code Δ
  ▷ : Code κ → Code κ
```

The constructors of Code Δ suffice for the specification of interesting examples of guarded recursive types such as streams. Nevertheless, it would not be complicated to add exponentials with constant domain and the box modality to the grammar. In addition, this grammar does not allow the possibility of defining nested inductive types.

3.2 Contexts

The well-formed contexts of **GTT** are built from the empty context, context extension, and context weakening. The last operation embeds ∅-contexts into κ-contexts. Notice that we are overloading the symbol ⇡, which is used for both type and context weakening.

```
data Ctx : ClockCtx → Set where
  • : ∀ {Δ} → Ctx Δ
  _,_ : ∀ {Δ} → Ctx Δ → Ty Δ → Ctx Δ
  ⇡ : Ctx ∅ → Ctx κ
```

3.3 Terms

The well-typed terms and substitutions of **GTT** are defined simultaneously. Terms include constructors for variables and substitutions.

```
data Tm : ∀ {Δ} → Ctx Δ → Ty Δ → Set where
  var : ∀ {Δ} (Γ : Ctx Δ) (A : Ty Δ) → Tm (Γ , A) A
  sub : ∀ {Δ} {Γ₁ Γ₂ : Ctx Δ} {A : Ty Δ} → Tm Γ₂ A → Sub Γ₁ Γ₂ → Tm Γ₁ A
```

We have lambda abstraction and application, plus the usual introduction and elimination rules for the unit types, products, coproducts, and guarded recursive types. Here we only show the typing rules associated to function types and guarded recursive types. The function

eval evaluates codes in Code Δ into endofunctors on Ty Δ. We use a categorical combinator app for application. We derive the conventional application, taking additionally an element in Tm Γ A and returning an inhabitant of Tm Γ B, in Section 3.4.

```
lambda : ∀ {Δ} {Γ : Ctx Δ} {A B : Ty Δ} → Tm (Γ , A) B → Tm Γ (A ⟶ B)
app : ∀ {Δ} {Γ : Ctx Δ} {A B : Ty Δ} → Tm Γ (A ⟶ B) → Tm (Γ , A) B
cons : ∀ {Δ} {Γ : Ctx Δ} (P : Code Δ) → Tm Γ (eval P (μ P)) → Tm Γ (μ P)
primrec : ∀ {Δ} (P : Code Δ) {Γ : Ctx Δ} {A : Ty Δ}
  → Tm Γ (eval P (μ P ⊠ A) ⟶ A) → Tm Γ (μ P ⟶ A)
```

The later modality is required to be an applicative functor, which means that we have terms next and ⊛. The delayed fixpoint combinator dfix [9] allows defining productive recursive programs. The usual fixpoint returning a term in A instead of ▷ A is derivable.

```
next : {Γ : Ctx κ} {A : Ty κ} → Tm Γ A → Tm Γ (▷ A)
_⊛_ : {Γ : Ctx κ} {A B : Ty κ} → Tm Γ (▷ (A ⟶ B)) → Tm Γ (▷ A) → Tm Γ (▷ B)
dfix : {Γ : Ctx κ} {A : Ty κ} → Tm Γ (▷ A ⟶ A) → Tm Γ (▷ A)
```

We have introduction and elimination rules for the □ modality. The rule box is the analogue in **GTT** of Atkey and McBride's rule for clock abstraction [8]. Notice that box can only be applied to terms of type A over a weakened context ↑ Γ. This is analogous to Atkey and McBride's side condition requiring the universally quantified clock variable to not appear freely in the context Γ. Similarly, the rule unbox corresponds to clock application. The operation force is used for removing occurrences of ▷ protected by the □ modality.

```
box : {Γ : Ctx ∅} {A : Ty κ} → Tm (↑ Γ) A → Tm Γ (□ A)
unbox : {Γ : Ctx ∅} {A : Ty κ} → Tm Γ (□ A) → Tm (↑ Γ) A
force : {Γ : Ctx ∅} {A : Ty κ} → Tm Γ (□ (▷ A)) → Tm Γ (□ A)
```

The introduction and elimination rules for type weakening say that elements of Tm Γ A can be embedded in Tm (↑ Γ) (↑ A) and vice versa.

```
up : {Γ : Ctx ∅} {A : Ty ∅} → Tm Γ A → Tm (↑ Γ) (↑ A)
down : {Γ : Ctx ∅} {A : Ty ∅} → Tm (↑ Γ) (↑ A) → Tm Γ A
```

Atkey and McBride assume the existence of certain type equalities [8]. Møgelberg, who works in a dependently typed setting, considers similar type isomorphisms [20]. In **GTT**, we follow the second approach. This means that we do not introduce an equivalence relation on types specifying which types should be considered equal, as in Chapman's object type theory [13]. Instead, we include additional term constructors corresponding to functions underlying the required type isomorphisms. For example, the clock irrelevance axiom formulated in our setting states that every ∅-type A is isomorphic to □ (↑ A). This is obtained by adding to Tm a constructor □const.

```
□const : {Γ : Ctx ∅} (A : Ty ∅) → Tm Γ (□ (↑ A) ⟶ A)
```

A function const□ A in the other direction is derivable. When defining definitional equality on terms, described in Section 3.5, we ask for □const and const□ to be each other inverses. The other type isomorphisms, listed in Appendix A are constructed in a similar way.

3.4 Substitutions

For substitutions, we need the canonical necessary operations [7, 13]: identity and composition of substitutions, the empty substitution, the extension with an additional term, and the projection which forgets the last term.

```
data Sub : ∀ {Δ} → Ctx Δ → Ctx Δ → Set where
  ε : ∀ {Δ} (Γ : Ctx Δ) → Sub Γ •
  id : ∀ {Δ} (Γ : Ctx Δ) → Sub Γ Γ
  _,_ : ∀ {Δ} {Γ₁ Γ₂ : Ctx Δ} {A : Ty Δ} → Sub Γ₁ Γ₂ → Tm Γ₁ A → Sub Γ₁ (Γ₂ , A)
  _∘_ : ∀ {Δ} {Γ₁ Γ₂ Γ₃ : Ctx Δ} → Sub Γ₂ Γ₃ → Sub Γ₁ Γ₂ → Sub Γ₁ Γ₃
  pr : ∀ {Δ} {Γ₁ Γ₂ : Ctx Δ} {A : Ty Δ} → Sub Γ₁ (Γ₂ , A) → Sub Γ₁ Γ₂
```

We also add rules for embedding substitutions between ∅-contexts into substitutions between κ-contexts and vice versa.

```
  up : {Γ₁ Γ₂ : Ctx ∅} → Sub Γ₁ Γ₂ → Sub (⇑ Γ₁) (⇑ Γ₂)
  down : {Γ₁ Γ₂ : Ctx ∅} → Sub (⇑ Γ₁) (⇑ Γ₂) → Sub Γ₁ Γ₂
```

In addition, we require the existence of two context isomorphisms. The context ⇑ • needs to be isomorphic to • and ⇑ (Γ , A) needs to be isomorphic to ⇑ Γ , ⇑ A. For both of them, we add a constructor representing the underlying functions.

```
  •⇑ : Sub • (⇑ •)
  ,⇑ : (Γ : Ctx ∅) (A : Ty ∅) → Sub (⇑ Γ , ⇑ A) (⇑ (Γ , A))
```

An element ⇑• in Sub (⇑ •) • is derivable. In the definitional equality on substitutions, we ask for •⇑ and ⇑• to be each other inverses. We proceed similarly with ,⇑.

Using the term constructor sub, we can derive a weakening operation for terms and the conventional application combinator.

```
wk : ∀ {Δ} {Γ : Ctx Δ} {A B : Ty Δ} → Tm Γ B → Tm (Γ , A) B
wk {Δ} {Γ} {A} x = sub x (pr (id (Γ , A)))

_$_ : ∀ {Δ} {Γ : Ctx Δ} {A B : Ty Δ} → Tm Γ (A ⟶ B) → Tm Γ A → Tm Γ B
_$_ {Δ} {Γ} f x = sub (app f) (id Γ , x)
```

3.5 Definitional equalities

Definitional equalities on terms and substitutions are defined simultaneously. Here we only discuss equality on terms and we refer to the formalization for the equality on substitutions.

```
data _∼_ : ∀ {Δ} {Γ : Ctx Δ} {A : Ty Δ} (t₁ t₂ : Tm Γ A) → Set where
```

The term equality includes rules for equivalence, congruence, and substitution. There also are β and η rules for the type formers. We only show the ones associated to the □ modality here. The rules state that box and unbox are each other's inverses.

```
□-β : ∀ {Γ} {A} (t : Tm (⇑ Γ) A) → unbox (box t) ∼ t
□-η : ∀ {Γ} {A} (t : Tm Γ (□ A)) → box (unbox t) ∼ t
```

We include definitional equalities stating that ▷ is an applicative functor w.r.t. the operations next and ⊛. Furthermore, the delayed fixpoint combinator dfix must satisfy its characteristic unfolding equation. We refer to Møgelberg's paper [20] for a complete list of the required definitional equalities for ▷ and □.

For the type isomorphisms, we require term equalities exhibiting that certain maps are mutual inverses. For example, we have the following two equalities about □const and const□:

```
const□const : ∀ {Γ} {A} (t : Tm Γ (□ (↑ A))) → const□ A $ (□const A $ t) ∼ t
□const□ : ∀ {Γ} {A} (t : Tm Γ A) → □const A $ (const□ A $ t) ∼ t
```

The last group of term equalities describes the relationship between the weakening operations up and down and other term constructors. Here we omit their description and we refer the interested reader to the Agda formalization.

3.6 Example: Streams

We give a taste of how to program with streams in **GTT**. Our implementation is based on Atkey and McBride's approach to coinductive types [8]. To define a type of streams, we first define guarded streams over a ∅-type A. It is the least fixpoint of the functor with code ℂ (↑ A) ⊠ ▷ I.

```
F : Ty ∅ → Code κ
F A = ℂ (↑ A) ⊠ ▷ I

g-Str : Ty ∅ → Ty κ
g-Str A = μ (F A)
```

The usual type of streams over A is then obtained by applying the □ modality to g-Str A.

```
Str : Ty ∅ → Ty ∅
Str A = □ (g-Str A)
```

We compute the head and tail of a stream using a function decons, which destructs an element of an inductive type. The term decons is the inverse of cons and it is derivable using primrec. Note that in both cases we need to use unbox, because of the application of the box modality in the definition of Str. For the tail, we also use box and force. The operations π_1 and π_2 are the projections associated to the product type former ⊠.

```
hd : {Γ : Ctx ∅} {A : Ty ∅} → Tm Γ (Str A) → Tm Γ A
hd xs = down (π₁ (decons (unbox xs)))


tl : {Γ : Ctx ∅} {A : Ty ∅} → Tm Γ (Str A) → Tm Γ (Str A)
tl xs = force (box (π₂ (decons (unbox xs))))
```

Given a **GTT** term a of type A, we can construct the constant guarded stream over a using the fixpoint combinator.

```
g-const-str : {Γ : Ctx ∅} {A : Ty ∅} → Tm Γ A → Tm (↑ Γ) (g-Str A)
g-const-str {Γ} {A} a = fix (lambda (cons (F A) [ wk (up a) & var (↑ Γ) (▷ (g-Str A)) ]))
```

The constant stream over a is obtained by boxing the guarded stream g-const a.

```
const-str : {Γ : Ctx ∅} {A : Ty ∅} → Tm Γ A → Tm Γ (Str A)
const-str a = box (g-const-str a)
```

In our Agda formalization, we also construct a function removing the elements at even indices out of a given stream, which is an example of a non-causal function.

4 Categorical Semantics

Next we give a categorical semantics for the calculus introduced in Section 3. We take inspiration from Møgelberg's model [20] in which a (simple) type in a clock context containing n clocks is interpreted as a presheaf in $\mathbf{Set}^{(\omega^n)^{\mathrm{op}}}$, where ω is the preorder of ordered natural numbers. In our model, we replace the category ω with the preorder of sizes. Moreover, we only consider the cases where either the clock context is empty or it contains exactly one clock. This means that a type in **GTT** is either interpreted as an element of Set or as a presheaf over Size. In this section, we show how to implement presheaves and how to model the fragment of **GTT** corresponding to simply typed lambda calculus. The interpretation of the guarded recursive and coinductive features of **GTT** is given in Section 5.

4.1 Presheaves

Presheaves are defined as a record PSh. The fields Obj and Mor represent the actions on objects and morphisms respectively, while MorId and MorComp are the functor laws. The type Size< (↑ i) contains sizes smaller or equal than i. In the type of MorId we use the reflexivity of the order on sizes, which means i : Size< (↑ i). In the type of MorComp we use transitivity.

```
record PSh : Set₁ where
  field
    Obj : Size → Set
    Mor : (i : Size) (j : Size< (↑ i)) → Obj i → Obj j
    MorId : {i : Size} {x : Obj i} → Mor i i x ≡ x
    MorComp : {i : Size} {j : Size< (↑ i)} {k : Size< (↑ j)} {x : Obj i}
      → Mor i k x ≡ Mor j k (Mor i j x)
```

Beside presheaves, we also need natural transformations. These are defined as a record NatTrans, consisting of a map nat-map and a proof of the usual commutativity requirement.

```
record NatTrans (P Q : PSh) : Set where
  field
    nat-map : (i : Size) → Obj P i → Obj Q i
    nat-com : (i : Size) (j : Size< (↑ i)) (x : Obj P i)
      → Mor Q i j (nat-map i x) ≡ nat-map j (Mor P i j x)
```

Products and sums of presheaves are defined pointwise. More precisely, we define the product as follows

```
ProdObj : Size → Set
ProdObj i = Obj P i × Obj Q i
```

The sum is defined similarly. The weakening type former ↑ of **GTT** is modeled using the constant presheaf, which we denote by Const. Function spaces are defined as the exponential of presheaves. The action on a size i of this presheaf consists of natural transformations restricted to sizes smaller or equal than i.

```
record ExpObj (P Q : PSh) (i : Size) : Set where
  field
    fun : (j : Size< (↑ i)) → Obj P j → Obj Q j
    funcom : (j : Size< (↑ i)) (k : Size< (↑ j)) (x : Obj P j)
      → Mor Q j k (fun j x) ≡ fun k (Mor P j k x)
```

All in all, we get an operation Exp : PSh → PSh → PSh.

4.2 Modelling Simple Types

To interpret the fragment of **GTT** corresponding to simply typed lambda calculus, we use Kripke semantics [18]. Semantic judgments, similar to their syntactic counterparts, are indexed by a clock context. We reuse the type ClockCtx for the semantic clock contexts. The semantic variable contexts are sets if the clock context is empty, and they are presheaves otherwise.

```
SemCtx : ClockCtx → Set₁
SemCtx ∅ = Set
SemCtx κ = PSh
```

Note that **GTT** is a simple type theory, thus types do not depend on contexts. For this reason, we define the type SemTy of semantic types in the same way as SemCtx.

The semantic terms of type A in context Γ are functions from Γ to A if the clock context is empty, and they are natural transformations between Γ and A otherwise.

```
SemTm : {Δ : ClockCtx} (Γ : SemCtx Δ) (A : SemTy Δ) → Set
SemTm {∅} Γ A = Γ → A
SemTm {κ} Γ A = NatTrans Γ A
```

Since **GTT** is a type theory with explicit substitutions, we must provide an interpretation for them as well. Semantic substitutions are maps between contexts and we define the type SemSub in the same way as SemTm. Definitional equality of semantic terms and substitutions is modeled as propositional equality.

We also need to provide a semantic version of the context operations, the simple type formers and the operations on substitutions. Since their definitions are standard, we do not discuss them in detail. For each of them, we need to make a case distinction based on the clock context. For example, the empty variable context ▪ is interpreted as the unit type in the clock context $\varnothing$, and it is interpreted as the terminal presheaf in the clock context κ. We use the operations on presheaves defined in Section 4.1 to interpret simple type formers. In the next section, we use the interpretation of function types, whose action of objects in the clock context κ is given by ExpObj. This is denoted by $A \Rightarrow B$ for semantic types A and B.

```
_⇒_ : ∀ {Δ} (A B : SemTy Δ) → SemTy Δ
_⇒_ {∅} A B = A → B
_⇒_ {κ} A B = Exp A B
```

5 Modelling Guarded Recursion

In this section, we add the required guarded recursive and coinductive features to the denotational semantics. We start by defining the semantic box modality together with its introduction and elimination rule. Then we construct the semantic later modality. We show how to define the fixpoint combinator and the force operation using sized types. In the end, we discuss how to model guarded recursive types.

5.1 Context Weakening and the Box Modality

Similarly to the weakening type former ↑, the weakening context former ↑ is modeled using the constant presheaf Const.

```
⇑ : SemCtx ∅ → SemCtx κ
⇑ Γ = Const Γ
```

Møgelberg models universal clock quantification by taking limits [20]. We define the semantic box modality ■ similarly: given a presheaf A, we take ■ A to be the limit of A. Formally, the limit of A is constructed as a record with two fields. The field ■cone is given by a family $f\ i$ in Obj $A\ i$ for each size i. The field ■com is a proof that the restriction of $f\ i$ to a size j smaller than i is equal to $f\ j$.

```
record ■ (A : SemTy κ) : SemTy ∅ where
  field
    ■cone : (i : Size) → Obj A i
    ■com : (i : Size) (j : Size< (↑ i)) → Mor A i j (■cone i) ≡ ■cone j
```

The semantic box modality is right adjoint to context weakening. In other words, the types Tm (⇑ Γ) A and Tm Γ (■ A) are isomorphic for all Γ and A. The function underlying the isomorphism is sem-box and its inverse is sem-unbox, modeling box and unbox respectively.

```
sem-box : (Γ : SemCtx ∅) (A : SemTy κ) (t : SemTm (⇑ Γ) A) → SemTm Γ (■ A)
■cone (sem-box Γ A t x) i      = nat-map t i x
■com (sem-box Γ A t x) i j     = nat-com t i j x

sem-unbox : (Γ : SemCtx ∅) (A : SemTy κ) (t : SemTm Γ (■ A)) → SemTm (⇑ Γ) A
nat-map (sem-unbox Γ A t) i x   = ■cone (t x) i
nat-com (sem-unbox Γ A t) i j x = ■com (t x) i j
```

5.2 The Later Modality

The semantic later modality is an operation ▶ on semantic κ-types. Recall that the later modality in the topos of trees [11] is defined as

$$
\begin{aligned}
(\blacktriangleright\ A)(0) &= \{*\} \\
(\blacktriangleright\ A)(n+1) &= A(n)
\end{aligned}
$$

We cannot directly replicate the latter in our setting since the preorder of sizes does not possess a least element. However, we know from [11] that the definition above is equivalent to $(\blacktriangleright\ A)(n) = \lim_{k<n} A(k)$. Adapting this to our setting leads to the following action of ▶ on objects:

```
record ▶ObjTry (A : SemTy κ) (i : Size) : Set where
  field
    ▶cone : (j : Size< i) → Obj A j
    ▶com : (j : Size< i) (k : Size< (↑ j)) → Mor A j k (▶cone j) ≡ ▶cone k
```

However, with this definition, we are unable to implement a terminating semantic fixpoint combinator. Later in this section we discuss why this is the case.

There are several ways to modify the above definition and implement a terminating fixpoint operation. A possible solution, which was suggested to us by Andrea Vezzosi, is using an inductive analogue of the predicate Size<, which we call SizeLt.

```
data SizeLt (i : Size) : Set where
  [_] : Size< i → SizeLt i
```

The type SizeLt is a mechanism for suspending computations. If we define a function f of type (j : SizeLt i) → Obj A j by pattern matching, then $f\ j$ does not reduce unless j is of the form [j'] for some j' : Size< i. This simple observation turns out to be essential for a terminating implementation of the fixpoint combinator.

From an inhabitant of SizeLt, we obtain an actual size by pattern matching.

```
size : ∀ {i} → SizeLt i → Size
size [ j ] = j
```

Furthermore, functions with domain SizeLt i can be specified using functions on Size< i.

```
elimLt : {A : Size → Set₁} {i : Size} → ((j : Size< i) → A j)
  → (j : SizeLt i) → A (size j)
elimLt f [ j ] = f j
```

We define the action on objects of the semantic later modality similarly to ▶ObjTry but with SizeLt in place of Size<. Before we do so, we introduce two auxiliary functions, which turn out to be handy when modeling guarded recursive types. The first is a function Later, which instead of a semantic κ-type, takes a sized type as its input. Its definition is the same as the type of the field ▶cone in ▶ObjTry but with Size< replaced by SizeLt.

```
Later : (Size → Set) → Size → Set
Later A i = (j : SizeLt i) → A (size j)
```

The second auxiliary function is LaterLim. It takes as input a sized type A together with a proof that it is antitone. Again its definition is the same as the type of the field ▶com in ▶ObjTry but with two applications of elimLt and Size< replaced by SizeLt.

```
LaterLim : (A : Size → Set) (m : (i : Size) (j : Size< (↑ i)) → A i → A j)
  → (i : Size) (x : Later A i) → Set
LaterLim A m i x = (j : SizeLt i)
  → elimLt (λ { j' → (k : SizeLt (↑ j'))
    → elimLt (λ k' → m j' k' (x [ j' ]) ≡ x [ k' ]) k }) j
```

Putting everything together, we obtain the following definition of the object part of the semantic later modality ▶. We refer to the Agda formalization for the action on the morphisms and the functor laws.

```
record ▶Obj (A : SemTy κ) (i : Size) : Set where
  field
    ▶cone : Later (Obj A) i
    ▶com : LaterLim (Obj A) (Mor A) i ▶cone
```

We omit the semantic equivalents of next and ⊛. To interpret the delayed fixpoint combinator dfix, we introduce an auxiliary term $\mathsf{sem\text{-}dfix}_1$, for which we only show how to construct the field ▶cone. This is defined using self-application.

```
sem-dfix₁ : (A : SemTy κ) (i : Size) → ExpObj (▶ A) A i → ▶Obj A i
▶cone (sem-dfix₁ A i f) [ j ] = fun f j (sem-dfix₁ A j f)
```

This definition is accepted by Agda's termination checker for two reasons:

- every recursive call is applied to a strictly smaller size;
- the usage of SizeLt in place of Size< in the definition of Later prevents indefinite unfolding, which would have happened if we used ▶ObjTry instead of ▶Obj.

In fact, if we would replace ▶Obj by ▶ObjTry as the return type of sem-dfix$_1$ while keeping the same definition (with j in place of [j]), we would obtain the following non-terminating sequence of reductions:

```
▶cone (sem-dfix₁ A i f)
     = λ j → fun f j (sem-dfix₁ A j f)
     = λ j → fun f j (record { ▶cone = λ k → fun f k (sem-dfix₁ A k f) ; ▶com = ...})
     = ...
```

The termination of the ▶com component of sem-dfix$_1$ additionally relies on the presence of elimLt in the definition of LaterLim.

The field nat-map of sem-dfix is defined using sem-dfix$_1$. We omit the nat-com component.

```
sem-dfix : (Γ : SemCtx κ) (A : SemTy κ) (f : SemTm Γ (▶ A ⇒ A)) → SemTm Γ (▶ A)
nat-map (sem-dfix Γ A f) i γ = sem-dfix₁ A i (nat-map f i γ)
```

Finally, we show how to interpret force. To this aim, we introduce an auxiliary function sem-force', which, given a type A and an inhabitant t of ■(▶ A), returns a term in ■ A. For the field ■cone of sem-force' A t, we are required to construct an element of Obj A i for each size i. Notice that ■cone t, when applied to a size i', gives a term t' of type ▶Obj A i'. Furthermore, the component ▶cone of t', when applied to a size j' smaller than i', returns a term of type Obj A j'. Hence, in order to construct the required inhabitant of Obj A i, it suffices to find a size j greater than i. An option for such a size j is ∞. The field ■com is defined in a similar way.

```
sem-force' : (A : SemTy κ) → ■ (▶ A) → ■ A
■cone (sem-force' A t) i = ▶cone (■cone t ∞) [ i ]
■com (sem-force' A t) i j = ▶com (■cone t ∞) [ i ] [ j ]
```

The semantic force operation follows immediately from sem-force'.

```
sem-force : (Γ : SemCtx ∅) (A : SemTy κ) (t : SemTm Γ (■ (▶ A))) → SemTm Γ (■ A)
sem-force Γ A t x = sem-force' A (t x)
```

5.3 Guarded Recursive Types

For semantic guarded recursive types, we introduce a type of semantic codes for functors. We cannot reutilize the syntactic grammar Code since the code for the constant functor should depend on SemTy rather than Ty. Instead we use the following definition.

```
data SemCode : ClockCtx → Set₁ where
  ℂ : ∀ {Δ} → SemTy Δ → SemCode Δ
  I : ∀ {Δ} → SemCode Δ
```

```
_⊞_ : ∀ {Δ} → SemCode Δ → SemCode Δ → SemCode Δ
_⊠_ : ∀ {Δ} → SemCode Δ → SemCode Δ → SemCode Δ
▶ : SemCode κ → SemCode κ
```

In the remainder of this section, we only discuss guarded recursive κ-types. The interpretation of μ in the clock context $\varnothing$ is standard and therefore omitted. Given a semantic code P, our goal is to construct the action on objects and morphisms of a presheaf mu-κ P.

A first naïve attempt would be to define the action on objects muObj P as the initial algebra of sem-eval P, where sem-eval evaluates a code as an endofunctor on SemTy κ. This means defining muObj P as an inductive type with one constructor taking in input sem-eval P (muObj P). This idea does not work, since muObj P is a sized type while sem-eval P expects a semantic κ-type.

Another possibility would be to define muObj P by induction on P. However, there is a problem when we arrive at the I constructor. In this case, we would like to make a recursive call to muObj applied to the original code P, which is unavailable at this point. We solve the issue by introducing an auxiliary inductive type family muObj', which depends on two codes instead of one. The first code is the original one used to define the guarded recursive type and we do induction on the second one. Then we define muObj P to be muObj' P P.

The constructors of muObj' P Q follow the structure of Q. If Q is a product we have a pairing constructor, if it is a sum we have the two injections. When Q is the code for the identity functor, we make a recursive call. For the ▶ case, we have a constructor later taking two arguments with the same types as the two fields of ▶Obj. Since LaterLim depends both on a sized type and a proof that it is antitone, we need to define muObj' mutually with its own proof of antitonicity muMor'. This construction works since Later and LaterLim take in input part of the data of a presheaf and they crucially do not depend on the functor laws.

```
mutual
  data muObj' (P : SemCode κ) : SemCode κ → Size → Set where
    const : {X : PSh} {i : Size} → Obj X i → muObj' P (ℂ X) i
    rec : ∀{i} → muObj' P P i → muObj' P I i
    _,_ : ∀{Q R i} → muObj' P Q i → muObj' P R i → muObj' P (Q ⊠ R) i
    in₁ : ∀{Q R i} → muObj' P Q i → muObj' P (Q ⊞ R) i
    in₂ : ∀{Q R i} → muObj' P R i → muObj' P (Q ⊞ R) i
    later : ∀{Q i} (x : Later (muObj' P Q) i)
      → LaterLim (muObj' P Q) (muMor' P Q) i x → muObj' P (▶ Q) i
```

```
muMor' : (P Q : SemCode κ) (i : Size) (j : Size< (↑ i)) → muObj' P Q i → muObj' P Q j
muMor' P (ℂ X) i j (const x) = const (Mor X i j x)
muMor' P I i j (rec x) = rec (muMor' P P i j x)
muMor' P (Q ⊠ R) i j (x , y) = muMor' P Q i j x , muMor' P R i j y
muMor' P (Q ⊞ R) i j (in₁ x) = in₁ (muMor' P Q i j x)
muMor' P (Q ⊞ R) i j (in₂ x) = in₂ (muMor' P R i j x)
muMor' P (▶ Q) i j (later x p) = later x (λ { [ k ] → p [ k ] })
```

We define muMor P to be muMor' P P. Since muMor preserves the identity and composition, we get a presheaf mu-κ P for each P. This is used to interpret μ in the clock context κ. We write mu for the interpretation of μ in a general clock context.

6 Soundness and Consistency

We now define the notion of interpretation of **GTT**. To interpret types, one must give a type of semantical types and a function mapping each syntactic type to its semantical counterpart. Similarly for contexts, terms, substitutions and definitional equalities. This leads to the following record where we only show the fields related to types.

```
record interpret-syntax : Set₂ where
  field
    semTy : ClockCtx → Set₁
    _⟦_⟧Ty : ∀ {Δ} → Ty Δ → semTy Δ
```

Now we prove **GTT** sound w.r.t. the categorical semantics. We only show the interpretation of the types whose semantics were explicitly discussed in Sections 4 and 5. Since syntactic types are defined mutually with codes, the interpretation of types ⟦_⟧type has to be defined simultaneously with the interpretation of codes ⟦_⟧code, which we omit here.

```
⟦_⟧type : ∀ {Δ} → Ty Δ → SemTy Δ
⟦ A ⟶ B ⟧type = ⟦ A ⟧type ⇒ ⟦ B ⟧type
⟦ ▷ A ⟧type = ► ⟦ A ⟧type
⟦ □ A ⟧type = ■ ⟦ A ⟧type
⟦ μ P ⟧type = mu ⟦ P ⟧code
```

Similarly, Ctx, Tm and Sub are mapped into SemCtx, SemTm and SemSub respectively. Definitional equality of terms is interpreted as Agda's propositional equality, the same for definitional equality of substitutions.

```
sem : interpret-syntax
semTy sem = SemTy
_⟦_⟧Ty sem = ⟦_⟧type
```

Using the interpetation sem, we can show that **GTT** is consistent, by which we mean that not every definitional equality is deducible. We first define a type bool : Ty ∅ as **1** ⊞ **1** and two terms TRUE and FALSE as in_1 tt and in_2 tt respectively, where in_1 and in_2 are the two constructors of ⊞. We say that **GTT** is consistent if TRUE and FALSE are not definitionally equal.

```
consistent : Set
consistent = TRUE ∼ FALSE → ⊥
```

This is proved by noticing that if TRUE were definitionally equal to FALSE, then their interpretations in sem would be equal. However, they are interpreted as inj_1 tt and inj_2 tt respectively, and those are unequal. Hence, **GTT** is consistent.

7 Conclusions and Future Work

We presented a simple type theory for guarded recursion that we called **GTT**. We formalized its syntax and semantics in Agda. Sized types were employed to interpret the characteristic features of guarded recursion. From this, we conclude that guarded recursion can be simulated using sized types. We greatly benefited from the fact that sized types constitute a native feature of Agda, so we were able to fully develop our theory inside a proof assistant.

This work can be extended in several different directions. Various type theories for guarded recursion in the literature include dependent types. Currently, there exist two disciplines for mixing dependent types with guarded recursion: delayed substitutions [12] and ticks [9]. It would be interesting to extend the syntax and semantics of **GTT** with dependent types following one of these two methods.

Abel and Vezzosi [5] formalized a simple type theory extended with the later modality in Agda. They focus on operational properties of the calculus and give a certified proof of strong normalization. As future work, we plan to investigate the metatheory and the dynamic behavior of **GTT** by formalizing a type checker and a normalization algorithm. This would also set up the basis for a usable implementation of **GTT**.

In **GTT** we restrict the least fixpoint operator μ to act exclusively on strictly positive functors. This restriction is already present in Atkey and McBride's calculus, which is aimed at encoding coinductive types using the later modality and it does not allow solving general guarded recursive domain equations. **GTT** is a variant of Atkey and McBride's type theory, therefore the similar restriction to stricly positive functors. From the formalization perspective, this restriction allows us to use Agda's inductive types to model the guarded recursive types of GTT, as shown in Section 5.3. In future work, we plan to extend **GTT** with the possibility of solving general guarded recursive domain equations. This could be done in two ways: extending the language with a universe, which allows the encoding of guarded recursive types as fixpoints in the universe as shown e.g. by Møgelberg [20]; or considering a μ type former which, besides stricly positive functors, also operates on functors where all variables are guarded by an occurrence of the later modality. In the second option we have to consider strictly positive functors to encode usual inductive types, such as the natural numbers.

Finally, we would like to understand whether sized types can be simulated using guarded recursion. This could either be done by modeling sized types in a topos of trees-like category [11, 19] or via the encoding of a type theory with sized types into a dependent type theory for guarded recursion such as Clocked Type Theory [9].

References

1. Martín Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy. Explicit Substitutions. *J. Funct. Program.*, 1(4):375–416, 1991. doi:10.1017/S0956796800000186.
2. Andreas Abel. MiniAgda: Integrating Sized and Dependent Types. In *Partiality and Recursion in Interactive Theorem Provers, PAR@ITP 2010, Edinburgh, UK, July 15, 2010*, pages 18–32, 2010. URL: http://www.easychair.org/publications/paper/51657.
3. Andreas Abel and James Chapman. Normalization by Evaluation in the Delay Monad: A Case Study for Coinduction via Copatterns and Sized Types. In *Proceedings 5th Workshop on Mathematically Structured Functional Programming, MSFP@ETAPS 2014, Grenoble, France, 12 April 2014.*, pages 51–67, 2014. doi:10.4204/EPTCS.153.4.
4. Andreas Abel, Brigitte Pientka, David Thibodeau, and Anton Setzer. Copatterns: Programming Infinite Structures by Observations. In *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy - January 23 - 25, 2013*, pages 27–38, 2013. doi:10.1145/2429069.2429075.
5. Andreas Abel and Andrea Vezzosi. A Formalized Proof of Strong Normalization for Guarded Recursive Types. In *Programming Languages and Systems - 12th Asian Symposium, APLAS 2014, Singapore, November 17-19, 2014, Proceedings*, pages 140–158, 2014. doi:10.1007/978-3-319-12736-1_8.
6. Andreas Abel, Andrea Vezzosi, and Théo Winterhalter. Normalization by Evaluation for Sized Dependent Types. *PACMPL*, 1(ICFP):33:1–33:30, 2017. doi:10.1145/3110277.

7 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, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 18–29, 2016. `doi:10.1145/2837614.2837638`.

8 Robert Atkey and Conor McBride. Productive Coprogramming with Guarded Recursion. In *ACM SIGPLAN International Conference on Functional Programming, ICFP'13, Boston, MA, USA - September 25 - 27, 2013*, pages 197–208, 2013.

9 Patrick Bahr, Hans Bugge Grathwohl, and Rasmus Ejlers Møgelberg. The Clocks are Ticking: No More Delays! In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12, 2017. `doi:10.1109/LICS.2017.8005097`.

10 Bruno Barras, Samuel Boutin, Cristina Cornes, Judicaël Courant, Jean-Christophe Filliatre, Eduardo Gimenez, Hugo Herbelin, Gerard Huet, Cesar Munoz, Chetan Murthy, et al. *The Coq Proof Assistant Reference Manual: Version 6.1.* PhD thesis, Inria, 1997.

11 Lars Birkedal, Rasmus Ejlers Møgelberg, Jan Schwinghammer, and Kristian Støvring. First Steps in Synthetic Guarded Domain Theory: Step-Indexing in the Topos of Trees. *Logical Methods in Computer Science*, 8(4), 2012. `doi:10.2168/LMCS-8(4:1)2012`.

12 Aleš Bizjak, Hans Bugge Grathwohl, Ranald Clouston, Rasmus E Møgelberg, and Lars Birkedal. Guarded Dependent Type Theory with Coinductive Types. In *International Conference on Foundations of Software Science and Computation Structures*, pages 20–35. Springer, 2016.

13 James Chapman. Type Theory Should Eat Itself. *Electr. Notes Theor. Comput. Sci.*, 228:21–36, 2009. `doi:10.1016/j.entcs.2008.12.114`.

14 Ranald Clouston, Ales Bizjak, Hans Bugge Grathwohl, and Lars Birkedal. Programming and Reasoning with Guarded Recursion for Coinductive Types. In *Foundations of Software Science and Computation Structures - 18th International Conference, FoSSaCS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, pages 407–421, 2015. `doi:10.1007/978-3-662-46678-0_26`.

15 Thierry Coquand. Infinite Objects in Type Theory. In *Types for Proofs and Programs, International Workshop TYPES'93, Nijmegen, The Netherlands, May 24-28, 1993, Selected Papers*, pages 62–78, 1993. `doi:10.1007/3-540-58085-9_72`.

16 Nils Anders Danielsson and Thorsten Altenkirch. Subtyping, Declaratively. In *Mathematics of Program Construction, 10th International Conference, MPC 2010, Québec City, Canada, June 21-23, 2010. Proceedings*, pages 100–118, 2010. `doi:10.1007/978-3-642-13321-3_8`.

17 John Hughes, Lars Pareto, and Amr Sabry. Proving the Correctness of Reactive Systems Using Sized Types. In *Conference Record of POPL'96: The 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Papers Presented at the Symposium, St. Petersburg Beach, Florida, USA, January 21-24, 1996*, pages 410–423, 1996. `doi:10.1145/237721.240882`.

18 Saunders MacLane and Ieke Moerdijk. *Sheaves in geometry and logic: A first introduction to topos theory*. Springer Science & Business Media, 1992.

19 Bassel Mannaa and Rasmus Ejlers Møgelberg. The Clocks They Are Adjunctions Denotational Semantics for Clocked Type Theory. In *3rd International Conference on Formal Structures for Computation and Deduction, FSCD 2018, July 9-12, 2018, Oxford, UK*, pages 23:1–23:17, 2018. `doi:10.4230/LIPIcs.FSCD.2018.23`.

20 Rasmus Ejlers Møgelberg. A Type Theory for Productive Coprogramming via Guarded Recursion. In *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 71:1–71:10, 2014. `doi:10.1145/2603088.2603132`.

21 Hiroshi Nakano. A Modality for Recursion. In *15th Annual IEEE Symposium on Logic in Computer Science, Santa Barbara, California, USA, June 26-29, 2000*, pages 255–266, 2000. `doi:10.1109/LICS.2000.855774`.

22 Ulf Norell. Dependently Typed Programming in Agda. In *Proceedings of TLDI'09: 2009 ACM SIGPLAN International Workshop on Types in Languages Design and Implementation, Savannah, GA, USA, January 24, 2009*, pages 1–2, 2009.

23 Jorge Luis Sacchini. Type-Based Productivity of Stream Definitions in the Calculus of Constructions. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 233–242, 2013. `doi:10.1109/LICS.2013.29`.

A Syntax of GTT

Contexts

$$\frac{}{- \vdash_\Delta} \qquad \frac{\Gamma \vdash_\Delta \quad \Gamma \vdash_\Delta A \text{ type}}{\Gamma, x : A \vdash_\Delta} \qquad \frac{\Gamma \vdash_\varnothing}{\uparrow\Gamma \vdash_\kappa}$$

Codes

$$\frac{}{\Gamma \vdash_\Delta \mathrm{I} \text{ code}} \qquad \frac{\Gamma \vdash_\Delta A \text{ type}}{\Gamma \vdash_\Delta A \text{ code}}$$

$$\frac{\Gamma \vdash_\Delta P \text{ code} \quad \Gamma \vdash_\Delta Q \text{ code}}{\Gamma \vdash_\Delta P \times Q \text{ code}} \qquad \frac{\Gamma \vdash_\Delta P \text{ code} \quad \Gamma \vdash_\Delta Q \text{ code}}{\Gamma \vdash_\Delta P + Q \text{ code}} \qquad \frac{\Gamma \vdash_\kappa P \text{ code}}{\Gamma \vdash_\kappa \triangleright P \text{ code}}$$

Types

$$\frac{}{\Gamma \vdash_\varnothing 1 \text{ type}} \qquad \frac{\Gamma \vdash_\Delta A \text{ type} \quad \Gamma \vdash_\Delta B \text{ type}}{\Gamma \vdash_\Delta A \times B \text{ type}}$$

$$\frac{\Gamma \vdash_\Delta A \text{ type} \quad \Gamma \vdash_\Delta B \text{ type}}{\Gamma \vdash_\Delta A + B \text{ type}} \qquad \frac{\Gamma \vdash_\Delta A \text{ type} \quad \Gamma \vdash_\Delta B \text{ type}}{\Gamma \vdash_\Delta A \to B \text{ type}}$$

$$\frac{\uparrow\Gamma \vdash_\kappa A \text{ type}}{\Gamma \vdash_\varnothing \Box A \text{ type}} \qquad \frac{\Gamma \vdash_\varnothing A \text{ type}}{\uparrow\Gamma \vdash_\kappa \uparrow A \text{ type}}$$

$$\frac{\Gamma \vdash_\kappa A \text{ type}}{\Gamma \vdash_\kappa \triangleright A \text{ type}} \qquad \frac{\Gamma \vdash_\Delta P \text{ code}}{\Gamma \vdash_\Delta \mu P \text{ type}}$$

Substitutions

$$\frac{\Gamma \vdash_\Delta}{\vdash_\Delta \varepsilon : \Gamma \to -} \qquad \frac{\Gamma \vdash_\Delta}{\vdash_\Delta \mathsf{id} : \Gamma \to \Gamma} \qquad \frac{\vdash_\Delta s : \Gamma_1 \to \Gamma_2, A}{\vdash_\Delta \mathsf{pr}\, s : \Gamma_1 \to \Gamma_2}$$

$$\frac{\vdash_\Delta s : \Gamma_1 \to \Gamma_2 \quad \Gamma_1 \vdash_\Delta t : A}{\vdash_\Delta s, t : \Gamma_1 \to \Gamma_2, A} \qquad \frac{\vdash_\Delta s_1 : \Gamma_1 \to \Gamma_2 \quad \vdash_\Delta s_2 : \Gamma_2 \to \Gamma_3}{\vdash_\Delta s_2 \circ s_1 : \Gamma_1 \to \Gamma_3}$$

$$\frac{\vdash_\varnothing s : \Gamma_1 \to \Gamma_2}{\vdash_\kappa \mathsf{up}\, s : \uparrow\Gamma_1 \to \uparrow\Gamma_2} \qquad \frac{\vdash_\kappa s : \uparrow\Gamma_1 \to \uparrow\Gamma_2}{\vdash_\varnothing \mathsf{down}\, s : \Gamma_1 \to \Gamma_2}$$

Terms

$$\frac{x : A \in \Gamma}{\Gamma \vdash_\Delta x : A} \qquad \frac{\vdash_\Delta s : \Gamma_1 \to \Gamma_2 \quad \Gamma_2 \vdash_\Delta t : A}{\Gamma_1 \vdash_\Delta t[s] : A}$$

$$\frac{\Gamma, x : A \vdash_\Delta t : B}{\Gamma \vdash_\Delta \lambda x.t : A \to B} \qquad \frac{\Gamma \vdash_\Delta f : A \to B \quad \Gamma \vdash_\Delta t : A}{\Gamma \vdash_\Delta f\,t : B}$$

$$\frac{}{\Gamma \vdash_\varnothing \mathsf{tt} : 1} \qquad \frac{\Gamma \vdash_\varnothing t : A}{\Gamma, x : 1 \vdash_\varnothing \mathsf{unitrec}\,t : A}$$

$$\frac{\Gamma \vdash_\Delta t : A \times B}{\Gamma \vdash_\Delta \pi_1 t : A} \qquad \frac{\Gamma \vdash_\Delta t : A \times B}{\Gamma \vdash_\Delta \pi_2 t : B}$$

$$\frac{\Gamma \vdash_\Delta t : A}{\Gamma \vdash_\Delta \mathsf{in}_1 t : A + B} \qquad \frac{\Gamma \vdash_\Delta t : B}{\Gamma \vdash_\Delta \mathsf{in}_2 t : A + B}$$

$$\frac{\Gamma \vdash_\Delta t_1 : A \quad \Gamma \vdash_\Delta t_2 : B}{\Gamma \vdash_\Delta (t_1, t_2) : A \times B} \qquad \frac{\Gamma, x : A \vdash_\Delta t_1 : C \quad \Gamma, y : B \vdash_\Delta t_2 : C}{\Gamma, z : A + B \vdash_\Delta \mathsf{plusrec}\,t_1\,t_2 : C}$$

$$\frac{\uparrow\Gamma \vdash_\kappa : t : A}{\Gamma \vdash_\varnothing \mathsf{box}\,t : \Box A} \qquad \frac{\Gamma \vdash_\varnothing : t : \Box A}{\uparrow\Gamma \vdash_\kappa \mathsf{unbox}\,t : A}$$

$$\frac{\Gamma \vdash_\varnothing t : A}{\uparrow\Gamma \vdash_\kappa \mathsf{up}\,t : \uparrow A} \qquad \frac{\uparrow\Gamma \vdash_\kappa t : \uparrow A}{\Gamma \vdash_\varnothing \mathsf{down}\,t : A}$$

$$\frac{\Gamma \vdash_\kappa t : A}{\Gamma \vdash_\kappa \mathsf{next}\,t : \triangleright A} \qquad \frac{\Gamma \vdash_\kappa f : \triangleright(A \to B) \quad \Gamma \vdash_\kappa t : \triangleright A}{\Gamma \vdash_\kappa f \circledast t : \triangleright B}$$

$$\frac{\Gamma \vdash_\kappa f : \triangleright A \to A}{\Gamma \vdash_\kappa \mathsf{dfix}\,f : \triangleright A} \qquad \frac{\Gamma \vdash_\varnothing t : \Box \triangleright A}{\Gamma \vdash_\varnothing \mathsf{force}\,t : \Box A}$$

$$\frac{\Gamma \vdash_\Delta t : F_P(\mu P)}{\Gamma \vdash_\Delta \mathsf{cons}\,t : \mu P} \qquad \frac{\Gamma \vdash f : F_P(\mu P \times A) \to A}{\Gamma \vdash_\Delta \mathsf{primrec}\,f : \mu P \to A}$$

where F_P is the evaluation of the code P into endofunctors on types, called eval P in Section 3. We omit the presentation of the definitional equalities of terms and substitutions, which can be found in our Agda formalization.

Type Isomorphisms

$$\Box(\uparrow A) \cong A \qquad \Box(A + B) \cong \Box A + \Box B \qquad \uparrow(A \to B) \cong \uparrow A \to \uparrow B \qquad \uparrow(\mu P) \cong \mu(\uparrow P)$$

Context Isomorphisms

$$- \cong \uparrow - \qquad \uparrow\Gamma, \uparrow A \cong \uparrow(\Gamma, A)$$

Sequence Types for Hereditary Permutators

Pierre Vial
Inria, Nantes, France
pierre.vial@inria.fr

Abstract

The invertible terms in Scott's model $\mathcal{D}_\infty$ are known as the hereditary permutators. Equivalently, they are terms which are invertible up to $\beta\eta$-conversion with respect to the composition of the λ-terms. Finding a type-theoretic characterization to the set of hereditary permutators was problem # 20 of TLCA list of problems. In 2008, Tatsuta proved that this was not possible with an inductive type system. Building on previous work, we use an infinitary intersection type system based on sequences (i.e., families of types indexed by integers) to characterize hereditary permutators with a unique type. This gives a positive answer to the problem in the coinductive case.

2012 ACM Subject Classification Theory of computation → Type theory

Keywords and phrases hereditary permutators, Böhm trees, intersection types, coinduction, ridigity, sequence types, non-idempotent intersection

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.33

Funding This research has been partially funded by the CoqHoTT ERC Grant 637339.

0 Introduction

The study of $\beta\eta$-invertible terms goes back to Curry and Feys [7], who showed that the only regular *combinators* having an inverse are of the form $\lambda x\, x_1 \ldots x_n.x\, x_{\sigma(1)} \ldots x_{\sigma(n)}$ with σ a permutation. Building on this work, Dezani [9] gave a characterization of the normal forms of all the invertible *normalizing* terms. This characterization was extended by Bergstra and Klop [3] for *any* term: $\beta\eta$-invertible terms were proved to have Böhm trees of a certain form, generalizing that given by Curry and Feys and suggesting to name them *hereditary permutators*.

On another hand, intersection types systems were introduced by Coppo and Dezani [6, 12] around 1980 (see [16] for a survey). They were extensively used to characterize various sets of terms having common semantic properties (including head, weak, strong normalization) in different calculi. Yet, hereditary permutators resisted such a characterization, so that the problem of finding a type system assigning a unique type to all hereditary permutators (and only to them) was inscribed in TLCA list of open problems by Dezani in 2006 (Problem # 20). Two years later, Tatsuta [14] proved that the set `HP` of hereditary head permutators is not recursively enumerable. This entails that `HP` cannot be characterized in an *inductive* type system.

However, in [17], using a *coinductive intersection type system* named system `S`, we characterized the so-called set of *hereditary head normalizing (HHN)* terms which is also a set of terms having Böhm trees of certain form (without the constant $\bot$), whereas this set was also proved not to be recursively enumerable by Tatsuta [13]. As in the finitary case, infinite types bring simpler semantic proofs of well-known theorems, e.g., system `S` helps proving that an asymptotic reduction strategy produces the infinitary normal form of a term when it exists. In this paper, we extend system `S` with a type constant characterizing the set of hereditary permutators and we thus give a positive answer to TLCA Problem # 20 in the coinductive case. This also proves that infinitary type systems may be used to characterize other sets of Böhm trees.

© Pierre Vial;
licensed under Creative Commons License CC-BY
4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019).
Editor: Herman Geuvers; Article No. 33; pp. 33:1–33:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Before properly starting the article, a few words should be said on system $\mathtt{S}$ and infinitary typing: intersections are represented by families of types indexed by sets K of integers $\geqslant 2$. These indexes are called *tracks*. Thus, system $\mathtt{S}$ is close to *non-idempotent* intersection, introduced by Gardner [10] and de Carvalho [5], for which $A \wedge A \neq A$. In the finite case, non-idempotency gives very simple proofs of normalization (see [4] for a survey). Tracks allow tracing occurrences of a given type in a derivation (*rigidity*) while ensuring syntax-direction, whereas having both is not possible when non-idempotent intersection is represented by lists or multisets. Rigidity is crucial in the infinitary case, because coinductive type grammars give birth to unsound derivations, e.g., the unsolvable term $\Omega := (\lambda x.x\,x)(\lambda x.x\,x)$ becomes typable. However, rigidity allows defining a validity criterion, called approximability, which brings back semantic soundness. This is why system $\mathtt{S}$ provides a good framework to characterize hereditary permutators.

Last, Tatsuta defines a type system with a family of type constants $\mathtt{ptyp}_d$ (with $d \in \mathbb{N}$) such that $t : \mathtt{ptyp}_d$ iff t is a hereditary permutator on d levels. Then, a term is a hereditary permutator iff $t : \mathtt{ptyp}_d$ is derivable for all $d \in \mathbb{N}$. However, given a hereditary permutator t, there is no explicit relation between the different typings $t : \mathtt{ptyp}_d$ when d ranges over $\mathbb{N}$. We reuse this idea here, but the notion of approximability hinted at above allows formally expressing the typing derivations concluding with $t : \mathtt{ptyp}_d$ as *extensions* of those concluding with $t : \mathtt{ptyp}_{d_0}$ with $d_0 < d$. Actually, we define a type constant $\mathtt{ptyp}$, which can be assigned to hereditary permutators and to them only, which is the “supremum” of all $\mathtt{ptyp}_d$ i.e., such that a typing $t : \mathtt{ptyp}$ is an extension of typings $t : \mathtt{ptyp}_d$ for all $d \in \mathbb{N}$.

Structure of the paper. We conclude this introduction with some technical background on hereditary permutators. Section 1 recalls some basic definitions about Böhm trees and the infinite λ-calculus, but also on system $\mathtt{S}$ and infinite types. In Section 2, we give a type-theoretic characterization of hereditary permutations in system $\mathtt{S}$. In Section 3, we introduce system $\mathtt{S}_{\mathtt{hp}}$, an extension of system $\mathtt{S}$, such that hereditary permutators have a unique type. The technical contributions of this paper are found mainly in Section 2 and 3.1.

Hereditary Permutators

Let $\mathcal{V}$ be a set of term variables. For all $n \in \mathbb{N}$, $\mathfrak{S}_n$ denotes the set of permutations of $\{1, \ldots, n\}$, $\to_{\mathtt{h}}$ denotes head reduction and the reflexive-transitive closure of a reduction $\to_{\mathcal{R}}$ is denoted $\to_{\mathcal{R}}^*$. To define hereditary permutators, we first consider *headed* hereditary permutators, i.e., hereditary permutators whose head variables have not been bound yet.

▶ **Definition 1.**

- *For all $x \in \mathcal{V}$, the sets $\mathtt{HP}(x)$ of **x-headed Hereditary Permutators (x-HP)** ($x \in \mathcal{V}$) are defined by mutual coinduction:*

$$\frac{h_1 \in \mathtt{HP}(x_1) \ \ldots \ h_n \in \mathtt{HP}(x_n) \quad (n \geqslant 0, \sigma \in \mathfrak{S}_n,\ x_i \neq x,\ x_i \text{ pairwise distinct}) \\ \text{and } h \to_{\mathtt{h}}^* \lambda x_1 \ldots x_n.x\, h_{\sigma(1)} \ldots h_{\sigma(n)}}{h \in \mathtt{HP}(x)}$$

- *A closed hereditary permutator, or simply, a **Hereditary Permutator (HP)** is a term of the form $h = \lambda x.h_0$ with $h_0 \in \mathtt{HP}(x)$ for some x.*

A headed hereditary permutator is the head reduct of a hereditary permutator applied to a variable.

▶ **Theorem 2** ([3]). *A λ-term t is a hereditary permutator iff t is invertible modulo $\beta\eta$-conversion for the operation $\cdot$ defined by $u \cdot v = \lambda x.u\,(v\,x)$, whose neutral element is $I = \lambda x.x$.*

Thus, u is invertible when there exists v such that $\lambda x.u(v\,x) =_{\beta\eta}= \lambda x.v(u\,x) =_{\beta\eta} I$. An extensive presentation of hereditary permutators and their properties is given in Chapter 21 of [2].

1 Infinite terms and types

In this section, we present Böhm trees (Chapter 10 of [2]) and the construction of one of the infinitary calculi introduced in [11]. See also [8, 1] for alternative presentations. We then present system $\mathtt{S}$, an infinitary intersection type system with a validity criterion (approximability) discarding unsound coinductive derivations and using sequences to represent intersection. Some more details can also be found in [17].

General notations. The set of finite words on $\mathbb{N}$ is denoted with $\mathbb{N}^*$, ε is the empty word, $a \cdot a'$ the concatenation of a and a'. The prefix order $\preceq$ is defined on $\mathbb{N}^*$ by $a \preceq a'$ if there is a_0 such that $a' = a \cdot a_0$, e.g., $2{\cdot}3 \preceq 2{\cdot}3{\cdot}0{\cdot}1$.

Intuitively, 0 is dedicated to the constructor λx, 1 is dedicated to the left-hand side of applications and all the $k \geqslant 2$ to the possibly multiple typings of the arguments of applications. This also explains why 0 and 1 will have a particular status in the definitions to come. For instance, the **applicative depth** $\mathtt{ad}(a)$ of $a \in \mathbb{N}^*$ is the number of nestings inside arguments, i.e., $\mathtt{ad}(a)$ is defined inductively by $\mathtt{ad}(\varepsilon) = 0$, $\mathtt{ad}(a \cdot k) = \mathtt{ad}(a)$ if $k = 0$ or $k = 1$ and $\mathtt{ad}(a \cdot k) = \mathtt{ad}(a) + 1$ if $k \geqslant 2$. The **collapse** is defined on $\mathbb{N}$ by $\overline{k} = \min(k, 2)$ and on $\mathbb{N}^*$ inductively by $\overline{\varepsilon} = \varepsilon$, $\overline{a \cdot k} = \overline{a} \cdot \overline{k}$, e.g., $\overline{7} = 2$, $\overline{1} = 1$ and $\overline{2{\cdot}3{\cdot}0{\cdot}1} = 2{\cdot}2{\cdot}0{\cdot}1$. These notions are straightforwardly extended to words of infinite length, e.g., 2^ω, which is the infinite repetition of 2.

1.1 Infinite Lambda Terms

The set Λ^∞ of infinitary λ-terms is coinductively defined by:

$$t, u \;:=\; x \in \mathcal{V} \parallel (\lambda x.t) \parallel (t\,u)$$

When there is no ambiguity, we usually just write $\lambda x.t$ and $t\,u_1 \ldots u_n$ instead of $(\lambda x.t)$ and $(\ldots(t\,u_1)\ldots u_n)$. If t is an infinitary term, then $\mathtt{supp}(t)$, the **support** of t (the set of positions in t) is defined in the usual way, i.e., coinductively, $\mathtt{supp}(x) = \{\varepsilon\}$, $\mathtt{supp}(\lambda x.t) = \{\varepsilon\} \cup 0 \cdot \mathtt{supp}(t)$ and $\mathtt{supp}(t\,u) = \{\varepsilon\} \cup 1 \cdot \mathtt{supp}(t) \cup 2 \cdot \mathtt{supp}(u)$. If $\overline{a} \in \mathtt{supp}(t)$, the subterm (resp. the constructor) of t at position $\overline{a}$ is denoted $t|_a$ (resp. $t(a)$), e.g., if $t = \lambda x.(x\,y)z$ and $a = 0 \cdot 1$ (resp. $a = 0 \cdot 4$), then $t|_a = x\,y$ and $t(a) = @$) (resp. $t|_a = t(a) = z$).

▶ **Definition 3** (001-Terms). *Let $t \in \Lambda^\infty$. Then t is a **001-term**, if, for all infinite branches γ in $\mathtt{supp}(t)$, $\mathtt{ad}(\gamma) = \infty$.*

Once again, the vocable "001-term" comes from [11]. For instance, the 001-term f^ω is formally defined as the tree such that $\mathtt{supp}(f^\omega) = \{2^n \mid n \in \mathbb{N}\} \cup \{2^n \cdot 1 \,|\, n \in \mathbb{N}\}$, $f^\omega(2^n) = @$ and $f^\omega(2^n \cdot 1) = f$ for all $n \in \mathbb{N}$. Its unique infinite branch is 2^ω (since all the finite prefixes of 2^ω are in $\mathtt{supp}(f^\omega)$), which satisfies $\mathtt{ad}(2^\omega) = \infty$. In contrast, the infinite term t defined by $t = t\,x$, so that $t = (((\ldots)x)x)x$, is *not* a 001-term: indeed, $\mathtt{supp}(t) = \{1^n \,|\, n \in \mathbb{N}\} \cup \{1^n \cdot 2 \,|\, n \in \mathbb{N}\}$, so $\mathtt{supp}(t)$ has the infinite branch 1^ω (this indicates a leftward infinite branch), which satisfies $\mathtt{ad}(1^\omega) = 0$ since 2 does not occur in 1^ω.

1.2 The computation of Böhm trees

The notation $t[u/x]$ denotes the term obtained from t by the capture-free substitution of the occurrences of x with u ([11] gives a formal definition in the infinitary calculus). The β-reduction $\to_\beta$ is obtained by the contextual closure of $(\lambda x.t)u \to_\beta t[u/x]$ and $t \xrightarrow{b}_\beta t'$ denotes the reduction of a redex at position b in t, e.g., $\lambda y.((\lambda x.x)u)v \xrightarrow{0\cdot 1}_\beta \lambda y.u\,v$. A **001-Normal Form (001-NF)** is a 001-term that does not contain a redex. A 001-term is **solvable** if $t \to_{\mathtt{h}}^* \lambda x_1 \dots x_p.x\, t_1 \dots t_q$, which is a head normal form (HNF) of **arity** p.

▶ **Definition 4** (Böhm tree of a term). *Let t be a 001-term.*
The Böhm tree $\mathtt{BT}(t)$ of t is coinductively defined by:
- $\mathtt{BT}(t) = \lambda x_1 \dots x_p.x\,\mathtt{BT}(t_1)\dots\mathtt{BT}(t_q)$ *if* $t \to_{\mathtt{h}}^* \lambda x_1 \dots x_p.x\, t_1 \dots t_q$.
- $\mathtt{BT}(t) = \bot$ *if t is unsolvable.*

For instance, $\mathtt{BT}(\Omega) = \bot$ where $\Omega = (\lambda x.x\,x)(\lambda x.x\,x)$ and $\mathtt{BT}(t) = t$ if t a 001-normal form. Definition 1 can be read as the specification of a set of terms whose Böhm trees have a particular form. Intuitively, the computation of Böhm trees is done by a possibly infinite series of head reductions at deeper and deeper levels. This corresponds to an asymptotic reduction strategy known as *hereditary head reduction*.

Some reduction paths are of infinite length but asymptotically produce a term.

▶ **Definition 5** (Productive reduction paths). *Let $t = t_0 \xrightarrow{b_0}_\beta t_1 \xrightarrow{b_1}_\beta t_2 \dots t_n \xrightarrow{b_n}_\beta t_{n+1} \dots$ be a reduction path of length $\ell \leqslant \omega$.*
Then, this reduction path is said to be ***productive*** *if either it is of finite length ($\ell \in \mathbb{N}$), or $\ell = \infty$ and $\mathtt{ad}(b_n)$ tends to infinity (recall that $\mathtt{ad}(\cdot)$ is applicative depth).*

A productive reduction path is called a *strongly converging reduction sequence* in [11], in which numerous examples are found. When $\mathtt{BT}(t)$ does not contain $\bot$, the hereditary head reduction strategy on a term t gives a particular case of productive path.

▶ **Lemma 6** (Limits of productive paths). *Let $t = t_0 \xrightarrow{b_0}_\beta t_1 \xrightarrow{b_1}_\beta t_2 \dots t_n \xrightarrow{b_n}_\beta t_{n+1} \dots$ be a productive reduction path of infinite length.*
Then, there is a 001-term t' such that, for every $d \geqslant 0$, there is $N \in \mathbb{N}$ such that, for all $n \geqslant N$, $\mathtt{supp}(t_n) \cap \{b \in \{0,1,2\}^* \mid \mathtt{ad}(b) \leqslant d\} = \mathtt{supp}(t') \cap \{b \in \{0,1,2\}^* \mid \mathtt{ad}(b) \leqslant d\}$.

The term t' in the statement of Lemma 6 is called the **limit** of the productive path. Intuitively, when t' is the limit of $(t_n)_{n\geqslant 0}$, then t' induces the same tree as t_n at fixed applicative depth after sufficiently many reduction steps. We then write $t \to_\beta^\infty t'$ if $t \to_\beta^* t'$ or t is the limit of a productive path starting at t. For instance, if $\Delta_f = \lambda x.f(x\,x)$, $\mathtt{Y}_f = \Delta_f\,\Delta_f$ (with $f \in \mathcal{V}$), then $\mathtt{Y}_f \xrightarrow{\varepsilon}_\beta f(\mathtt{Y}_f)$, which gives the productive path $\mathtt{Y}_f \xrightarrow{\varepsilon}_\beta f(\mathtt{Y}_f) \xrightarrow{2}_\beta \dots f^n(\mathtt{Y}_f) \xrightarrow{2^n}_\beta f^{n+1}(\mathtt{Y}_f)\dots$ since $\mathtt{ad}(2^n) \longrightarrow \infty$. The limit of this path – which implements hereditary head reduction on $\mathtt{Y}_f$ – is f^ω, i.e., $\mathtt{Y}_f \to_\beta^\infty f^\omega$ and also $\mathtt{BT}(\mathtt{Y}_f) = f^\omega$.

A 001-term t is said to be **infinitary weakly normalizing ($\mathbf{WN}_\infty$)** if there is a 001-NF t' such that $t \to_\beta^\infty t'$. It turns out that t is WN_∞ iff its Böhm tree does not contain $\bot$. The result is proved in [11] in a syntactical way, but we give a semantic proof of this fact in [17].

1.3 System $\mathtt{S}$ (sequential intersection)

A **sequence** of elements of a set X is a family $(x_k)_{k\in K}$ with $K \subseteq \mathbb{N} \setminus \{0,1\}$. In this case, if $k_0 \in K$, x_{k_0} is the element of $(x_k)_{k\in K}$ **on track** k. We often write $(k{\cdot}x_k)_{k\in K}$ for $(x_k)_{k\in K}$, which, for instance, allows us to denote by $(2\cdot a, 4\cdot b, 5\cdot a)$ or $(4\cdot b, 2\cdot a, 5\cdot a)$ the sequence

$(x_k)_{k\in K}$ with $K = \{2, 4, 5\}$, $x_2 = x_5 = a$ and $x_4 = b$. In this sequence, the element on track 4 is b. Sequences come along with a **disjoint union** operator, denoted $\uplus$. Let $(x_k)_{k\in K}$ and $(x'_k)_{k\in K'}$ be two sequences:

- If $K \cap K' = \emptyset$, then $(x_k)_{k\in K} \uplus (x'_k)_{k\in K'}$ is $(x'')_{k\in K''}$ with $K'' = K \cup K'$ and $x''_k = x_k$ when $k \in K$ and $x''_k = x'_k$ when $k \in K'$.
- If $K \cap K' \neq \emptyset$, $(x_k)_{k\in K} \uplus (x'_k)_{k\in K'}$ is not defined.

The operator $\uplus$ is partial, associative and commutative.

Let $\mathcal{O}$ be a set of type atoms o. The set of $\mathtt{S}$-types is coinductively defined by:

$$T,\, S_k \quad ::= \quad o \in \mathcal{O} \parallel (S_k)_{k\in K} \to T$$

A sequence of types $(S_k)_{k\in K}$ is called a **sequence type** and it represents an intersection of types. The types of system $\mathtt{S}$ collapse on usual non-idempotent intersection types built on multisets [4], e.g., the $\mathtt{S}$-types $(2\cdot o, 3\cdot o', 4\cdot o) \to o$ and $(2\cdot o', 8\cdot o, 9\cdot o) \to o$ collapse on $[o, o, o'] \to o$. The system is *strict* [12, 15, 16] since intersections occur only on left-hand sides of arrows. The **domain** and **codomain** of an arrow type are defined by $\mathtt{dom}((S_k)_{k\in K} \to T) = (S_k)_{k\in K}$ and $\mathtt{codom}((S_k)_{k\in K} \to T) = T$. The **arity** of a type is coinductively defined by $\mathtt{ar}(o) = 0$ and $\mathtt{ar}((S_k)_{k\in K} \to T) = \mathtt{ar}(T) + 1$. For instance, if T is defined by $T = (2\cdot o) \to T = (2\cdot o) \to (2\cdot o) \to \ldots$, then $\mathtt{ar}(T) = \infty$.

The **support** of a type or a sequence type U is coinductively defined by $\mathtt{supp}(o) = \{\varepsilon\}$, $\mathtt{supp}((S_k)_{k\in K}) = \cup_{k\in K} k\cdot \mathtt{supp}(S_k)$ and $\mathtt{supp}((S_k)_{k\in K} \to T) = \{\varepsilon\} \cup \mathtt{supp}((S_k)_{k\in K}) \cup 1\cdot \mathtt{supp}(T)$. Since $1 \notin K$ by convention, this definition is correct. If $c \in \mathtt{supp}(U)$, then $U|_c$ denotes the type or sequence type rooted at position c in U ($U|_c$ is a type when U is a type or $c \neq \varepsilon$). For instance, if $U = (2\cdot o) \to (2\cdot o, 3\cdot o') \to o$ and $c = 1$, then $U|_c = (2\cdot o, 3\cdot o') \to o$. Likewise, $U(c)$ denotes the type constructor ($o \in \mathcal{O}$ or $\to$) at position c. With the same example, $U(\varepsilon) = U(1) = \to$, $U(2) = o$ and $U(1\cdot 3) = o'$

A **001-type** is a $\mathtt{S}$-type T such that, for all $c \in \mathtt{supp}(T)$, $\mathtt{ar}(T|_c) < \infty$, where $T|_c$ is the subtree rooted at c in T ($T|_c$ is a type). The **target type** $\mathtt{targ}(T)$ of a 001-type S is *inductively* defined by $\mathtt{targ}(o) = o$ and $\mathtt{targ}((S_k)_{k\in K} \to T) = \mathtt{targ}(T)$.

An $\mathtt{S}$-context C (or D) is a total function from $\mathcal{V}$ to the set of $\mathtt{S}$-types. The operator $\uplus$ is extended point-wise. An $\mathtt{S}$-judgment is a triple $C \vdash t : T$, where C, t and T are respectively an $\mathtt{S}$-context, a 001-term and an $\mathtt{S}$-type. A **sequence judgment** is a sequence of judgments $(C_k \vdash t : T_k)_{k\in K}$ with $K \subseteq \mathbb{N} \setminus \{0, 1\}$. For instance, if $8 \in K$, the judgment $C_8 \vdash t : T_8$ is specified on track 8. The set of $\mathtt{S}$-derivations is defined coinductively by:

$$\frac{}{x : (k\cdot T) \vdash x : T}\ \mathtt{ax} \qquad \frac{C; x : (S_k)_{k\in K} \vdash t : T}{C \vdash \lambda x.t : (S_k)_{k\in K} \to T}\ \mathtt{abs}$$

$$\frac{C \vdash t : (S_k)_{k\in K} \to T \qquad (D_k \vdash u : S_k)_{k\in K}}{C \uplus (\uplus_{k\in K} D_k) \vdash t\, u : T}\ \mathtt{app}$$

In $\mathtt{app}$, K may be empty, and then u is untyped. We call $\mathtt{S}_0$, the restriction of system $\mathtt{S}$ to finite types and contexts, but allowing infinite terms. The derivation $P_{\mathtt{ex}}$ below is in $\mathtt{S}_0$.

Let P be a $\mathtt{S}$-derivation typing a term t. The **support** of P is the set of positions of judgments inside P defined in the expected way: 0 to visit the premise of an $\mathtt{abs}$-rule, 1 to visit the left-hand side of an $\mathtt{app}$-rule and $k \geqslant 2$ to visit an argument judgment on track k on the right-hand side of the $\mathtt{app}$-rule. Thus, if $a \in \mathtt{supp}(P)$, $P(a)$, which denotes the judgment at position a in P, types the subterm $t|_a$. We denote the type and the context of $P(a)$ by $\mathtt{T}^P(a)$ and $\mathtt{C}^P(a)$, so that $P(a) = \mathtt{C}^P(a) \vdash t|_a : \mathtt{T}^P(a)$. Moreover:

- If $a \in \mathtt{supp}(P)$ and $c \in \mathtt{supp}(\mathtt{T}^P(a))$, then the pair (a, c) is a **right biposition** of P and $P(a, c)$ denotes $\mathtt{T}^P(a)(c)$, which is a type constructor ($o \in \mathcal{O}$ or $\to$).

- If $a \in \mathtt{supp}(P)$, $x \in \mathcal{V}$ and $k \cdot c \in \mathtt{supp}(\mathtt{C}^P(a)(x))$, then the triple $(a, x, k \cdot c)$ is a **left biposition** in P and $P(a, x, c)$ denotes $\mathtt{C}^P(a)(x)(c)$.

The set of bipositions of P is called the **bisupport** of P and is denoted by $\mathtt{bisupp}(P)$. An $\mathtt{S}$-derivation P is **finite**, i.e., is a derivation of system $\mathtt{S}_0$, iff $\mathtt{bisupp}(P)$ is a finite set. If $a \in \mathtt{supp}(P)$ and $t(a) = x$, $P(a)$ is an $\mathtt{ax}$-rule and $\mathtt{C}^P(a) = x : (k \cdot \mathtt{T}^P(a))$.

▶ **Example 7.** In the derivation, the notation [7] indicates that the judgment $f : 2{\cdot}(\,)\to o \vdash f^\omega : o$ is on track 7.

$$P_{\mathtt{ex}} = \cfrac{\cfrac{}{x : (2{\cdot}(7 \cdot o) \to o') \vdash x : (7 \cdot o) \to o'}\,\mathtt{ax} \qquad \cfrac{\cfrac{}{f : (2{\cdot}(\,) \to o) \vdash f : (\,) \to o}\,\mathtt{ax}}{f : (2 \cdot (\,) \to o) \vdash f^\omega : o\ [7]}\,\mathtt{app}}{\cfrac{x : (2 \cdot (7 \cdot o) \to o'), f : (2 \cdot (\,) \to o) \vdash x\, f^\omega : o'}{x : (2 \cdot (7 \cdot o) \to o') \vdash \lambda f.x\, f^\omega : (2 \cdot (\,) \to o) \to o'}\,\mathtt{abs}}\,\mathtt{app}$$

We have $\mathtt{supp}(P_{\mathtt{ex}}) = \{\varepsilon, 0, 0{\cdot}1, 0{\cdot}7, 0{\cdot}7{\cdot}1\}$. Remark how f^ω is typed in the derivation using the type $(\,) \to o$. We have $P_{\mathtt{ex}}(0{\cdot}7{\cdot}1) = f : (2{\cdot}(\,)\to o) \vdash f : (\,)\to o$ and $\mathtt{T}^{P_{\mathtt{ex}}}(0{\cdot}1) = (7{\cdot}o)\to o$. Since $\mathtt{supp}(\mathtt{T}^{P_{\mathtt{ex}}}) = \{\varepsilon, 7, 1\}$ and $\mathtt{T}^{P_{\mathtt{ex}}}(\varepsilon) = \to$, $\mathtt{T}^{P_{\mathtt{ex}}}(7) = o$ and $\mathtt{T}^{P_{\mathtt{ex}}}(1) = o'$, we have $(0{\cdot}1, \varepsilon), (0{\cdot}1, 7), (0{\cdot}1, 1) \in \mathtt{bisupp}(P_{\mathtt{ex}})$ and $P_{\mathtt{ex}}(0{\cdot}1, \varepsilon) = \to$, $P_{\mathtt{ex}}(0{\cdot}1, 7) = o$, $P_{\mathtt{ex}}(0{\cdot}1, 1) = o'$. Likewise, $\mathtt{T}^{P_{\mathtt{ex}}}(0{\cdot}7) = o$ and $\mathtt{C}^{P_{\mathtt{ex}}}(0) = x : (2{\cdot}(7{\cdot}o)\to o), f : (2{\cdot}(\,)\to o)$, so that, e.g., $(0, x, 2{\cdot}7), (0, f, 2) \in \mathtt{bisupp}(P_{\mathtt{ex}})$, $P_{\mathtt{ex}}(0, x, 2{\cdot}7) = o$, $P_{\mathtt{ex}}(0, f, 2) = \to$.

A derivation P is quantitative when the context is computable from the axiom rules:

▶ **Definition 8** (Quantitative derivation). *Let P be a $\mathtt{S}$-derivation. Then P is **quantitative** if, for all $a \in \mathtt{supp}(P)$, $x \in \mathcal{V}$, $k \in \mathbb{N} \setminus \{0, 1\}$ such that $(a, x, k) \in \mathtt{bisupp}(P)$, there is $a_0 \succeq a$ such that $P(a_0) = x : (k \cdot S) \vdash x : S$.*

Observe that, in a quantitative derivation $P \rhd C \vdash t : T$, if $C(x) = (k \cdot S)$ i.e., x is assigned a singleton sequence type, then there is exactly one $\mathtt{ax}$-rule typing x (we use this in the proof of Claim 21).

An example of non-quantitative derivations is given by the family $(P_k)_{k \geqslant 2}$ defined by:

$$P_k = \cfrac{\cfrac{}{f : (k{\cdot}(2{\cdot}o)\to o) \vdash f : (2{\cdot}o)\to o}\,\mathtt{ax} \qquad P_{k+1} \rhd f : (\ell{\cdot}(2{\cdot}o)\to o)_{\ell \geqslant k+1}, x{:}(2{\cdot}o) \vdash f^\omega : o\ [2]}{f : (\ell{\cdot}(2{\cdot}o)\to o)_{\ell \geqslant k}, x : (2{\cdot}o) \vdash f^\omega : o}\,\mathtt{app}$$

The P_k type f^ω with o but they assign a non-empty sequence type to $x \notin \mathtt{fv}(f^\omega)$: this is why there are not quantitative. Indeed, $(\varepsilon, x, 2) \in \mathtt{bisupp}(P_k)$, but there is no $a_0 \in \mathtt{supp}(P_k)$ such that $P_k(a_0) = x : (2{\cdot}o) \vdash x : o$. Remark how the infinite branch of f^ω is used to assign a type to x whereas it does not occur in the subject. In contrast, if t is a finite λ-term, every derivation typing t is quantitative. However, Lemma 16 below states that quantitativity is a sufficient condition for soundness for normal forms.

1.4 Approximability

▶ **Definition 9** (Approximation). *Let P_0 and P be two $\mathtt{S}$-derivations typing the same term t. Then P_0 is an **approximation** of P if $\mathtt{bisupp}(P_0) \subseteq \mathtt{bisupp}(P)$ and, for all $\mathtt{p} \in \mathtt{bisupp}(P_0)$, $P_0(\mathtt{p}) = P(\mathtt{p})$. When this holds, we write $P_0 \leqslant P$.*

Intuitively, $P_0 \leqslant P$ if the derivation P_0 can be obtained from the derivation P by erasing some symbols inside P. For instance, let:

$$P^0_{\mathtt{ex}} = \cfrac{\cfrac{\overline{x : (2 \cdot () \to o') \vdash x : () \to o'}\ \mathtt{ax}}{x : (2 \cdot () \to o') \vdash x\, f^\omega : o}\ \mathtt{app}}{x : (2 \cdot () \to o') \vdash \lambda f.x\, f^\omega : () \to o'}\ \mathtt{abs}$$

Then $P^0_{\mathtt{ex}} \leqslant P_{\mathtt{ex}}$, since $P^0_{\mathtt{ex}}$ has been obtained from $P_{\mathtt{ex}}$ by erasing all typing information on f^ω. Indeed, we check that $\mathtt{supp}(P^0_{\mathtt{ex}}) \subseteq \mathtt{supp}(P_{\mathtt{ex}})$, $\mathtt{bisupp}(P^0_{\mathtt{ex}}) \subseteq \mathtt{bisupp}(P_{\mathtt{ex}})$ and $P^0_{\mathtt{ex}}(\mathtt{p}) = P_{\mathtt{ex}}(\mathtt{p})$ for all $\mathtt{p} \in \mathtt{bisupp}(P^0_{\mathtt{ex}})$.

The relation $\leqslant$ is an order. There are $\mathtt{S}$-derivations P that do not have finite approximations, e.g., any derivation typing Ω (see [17] for an example), but these derivation are *unsound*: they do not ensure any form of finitary or infinitary normalization. In contrast, a *finite* derivation is sound.

To retrieve validity, we must specify that infinitary derivations should be obtained as asymptotic extensions of *finite* derivations:

▶ **Definition 10** (Approximability). *Let P be a $\mathtt{S}$-derivation. Then P is* ***approximable*** *if P is the supremum of its finite approximations i.e., if, for all finite sets $B \subseteq \mathtt{bisupp}(P)$, there is a finite derivation P_0 such that $P_0 \leqslant P$ and $B \subseteq \mathtt{bisupp}(P_0)$.*

A term that is in the conclusion of an approximable derivation is said to be **approximably typable**. Quantitativity is of course a necessary condition for approximability, and types of infinite arity are unsound:

▶ **Lemma 11.** *If P is approximable, then P is quantitative and contains only 001-types.*

1.5 Soundness and completeness for system S

The main characterization theorem of system $\mathtt{S}$ states the equivalence between infinitary weak normalization and typability: more precisely, t is WN_∞ iff there is an unforgetful and approximable $\mathtt{S}$-derivation typing P. This characterization is proved by the propositions below, that we will also use in this article. One recognizes usual properties that are expected from an intersection type system (subject reduction, expansion, typing of the normal forms), except that they pertain to infinitary objects and computations.

▶ **Proposition 12** (Infinitary subject reduction). *If $P \rhd C \vdash t : T$ is approximable and $t \to^\infty_\beta t'$, then there is an approximable derivation $P' \rhd C \vdash t' : T$.*

If a term is approximably typable, then, in particular, it is *finitely* typable, so that it is HN, as for usual, inductive intersection type systems:

▶ **Lemma 13** (Approximability and Head Normalization). *If $P \rhd C \vdash t : T$ is approximable, then t is head normalizing.*

Approximable $\mathtt{S}$-derivations ensure only *head* normalization because of the empty sequence $()$, which allows us to leave an argument untyped. For instance, if x is assigned $() \to o$, then $x\,t$ is typed with o for any term t. To ensure WN_∞, one needs to control the occurrences of $()$: by definition, the empty sequence type $()$ occur negatively in $() \to T$ (base case), $()$ occurs negatively (resp. positively) in $(S_k)_{k \in K} \to T$ if it occurs negatively (resp. positively) in T or positively (resp. negatively) in one of the S_k (inductive case).

▶ **Definition 14** (Unforgetfulness). *Let $P \rhd C \vdash t : T$ be a derivation. Then P is unforgetful when $(\,)$ does not occur negatively in C and does not occur positively in T.*

In particular, when $(\,)$ does not occur in C nor in T, then P is unforgetful.

▶ **Proposition 15** (Correctness for system $\mathtt{S}$). *If $P \rhd C \vdash t : T$ is approximable and unforgetful, then t is infinitary weakly normalizing.*

Completeness for infinitary normal forms – i.e., the fact that they are approximably and unforgetfully typable – is proved in two steps: one shows that every *quantitative* derivation typing a normal form is approximable (this is not true for non-normal forms). Since one finds quantitative unforgetful derivations for each normal form, one concludes:

▶ **Lemma 16** (Completeness for Normal Forms). *Let t be a NF_∞.*

- *If $P \rhd C \vdash t : T$ and P is quantitative, then P is approximable.*
- *t is approximably typable by derivation P such that $\mathtt{supp}(P) = \mathtt{supp}(t)$.*

Subject expansion holds for productive reduction paths:

▶ **Proposition 17** (Infinitary subject expansion). *If $t \to_\beta^\infty t'$ and $P' \rhd C \vdash t' : T$ is approximable, then there is an approximable derivation concluding with $C \vdash t : T$.*

From Lemma 16 and Proposition 17, one concludes that every infinitary weakly normalizing term is approximably and unforgetfully typable.

Last, Propositions 12 and 17 entail:

▶ **Lemma 18.** *If t is WN_∞, then t and $\mathtt{BT}(t)$ have the same approximable typings.*

2 Characterizing hereditary permutators

We now want to define the *permutator pairs* (S, T) (with S, T types of system $\mathtt{S}$) so that the judgments of the form $x : (2 \cdot S) \vdash t : T$ characterize the x-HP (i.e., there is an approximable $P \rhd x : (2 \cdot S) \vdash t : T$ iff t is an x-HP). Informally, if $h = \lambda x_1 \ldots x_n . x\, h_{\sigma(1)} \ldots h_{\sigma(n)}$ and h is typed with a type of arity n and $x_1, \ldots, x_n$ are the respective head variables of $h_1, \ldots, h_n$, then we have:

- Type of h = (type of x_1) $\to \ldots \to$ (type of x_n) $\to o$ $\quad(\mathtt{eq}_1)$
- Type of x = (type of $h_{\sigma(1)}$) $\to \ldots \to$ (type of $h_{\sigma(n)}$) $\to o$ $\quad(\mathtt{eq}_2)$

Since $x_1, \ldots,\ x_n$ are the respective head variables of the headed hereditary permutators $h_1, \ldots, h_n$, the equations $(\mathtt{eq}_1)$ and $(\mathtt{eq}_2)$, which are the golden thread of the proofs to come in the remainder of the paper, suggest the following coinductive definition:

▶ **Definition 19** (Permutators pairs).

- *When o ranges over $\mathcal{O}$ (the set of type atoms), the set $\mathtt{PP}(o)$ of **o-permutator pairs** (S, T), where S and T are $\mathtt{S}$-types, is defined by mutual coinduction:*

$$\frac{(S_1, T_1) \in \mathtt{PP}(o_1), \ldots,\ (S_n, T_n) \in \mathtt{PP}(o_n) \qquad o_1, \ldots, o_n, o \text{ pairwise distinct} \qquad \sigma \in \mathfrak{S}_n}{((2 \cdot T_{\sigma(1)}) \to \ldots \to (2 \cdot T_{\sigma(n)}) \to o, (2 \cdot S_1) \to \ldots (2 \cdot S_n) \to o) \in \mathtt{PP}(o)}$$

- *A pair $(S, T) \in \mathtt{PP}(o)$ is said to be **proper**, if, for all $o' \in \mathcal{O}$, o' occurs at most once in S and in T. The set of proper o-permutator pairs is denoted $\mathtt{PPP}(o)$.*

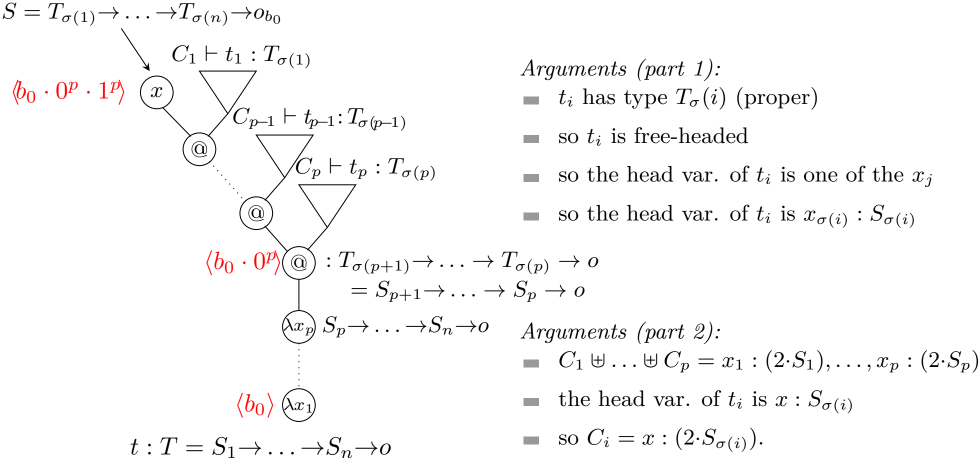


Figure 1 Hereditary permutators and permutator pairs.

Actually, we could allow other tracks than 2 in the definition (e.g., $T = (\ell_1 \cdot S_1) \to \ldots \to (\ell_n \cdot S_n) \to o$ would be fine), but it is more convenient to consider this restriction, so that we are relieved of the care of specifying the values of $\ell_1, \ldots, \ell_n$.

The condition of properness is here to ensure that every term variable occurs at exactly one level deeper than its binder and to distinguish them from one another: it is a key point of the proof of Claim 21, because two distinct variables will have types with distinct targets.

The first implication of the characterization is quite natural to prove:

▷ Claim 20 (From hereditary permutators to permutator pairs). Let $y \in \mathcal{V}$ and t be a y-head hereditary permutator. Then there is an approximable $\mathtt{S}$-derivation P and a permutator pair (S, T) such that $P \rhd y : (2 \cdot S) \vdash h : T$.

Proof. We skip the proof of this property. Observe that Definition 19 is designed so that it holds. The converse claim (Claim 21) is more difficult to prove and requires to be carefully verified. ◁

▷ Claim 21 (From permutator pairs to hereditary permutators). Let $t \in \Lambda^{001}$ be a 001-normal form and (S, T) a permutator pair and P a quantitative $\mathtt{S}$-derivation typing t.

- If $P \rhd \vdash t : (2 \cdot S) \to T$, then t is a hereditary permutator.
- If $P \rhd x : (2 \cdot S) \vdash T$, then t is a x-headed hereditary permutator.

In both cases, $\mathtt{supp}(P) = \mathtt{supp}(t)$.

Proof. The proof uses the following observation: let us say that a HNF is **free-headed** when its head variable is free. If (S, T) is a *proper* permutator pair and $t = \lambda x_1 \ldots x_p.x_i\, t_1 \ldots t_q$ (with $1 \leqslant i \leqslant p$) is a HNF which is not free headed, then t cannot have the types S and T, since the target of the type of x_i appears twice in the type of t.

We now start the proof, whose main stages are summarized in Fig. 1, in which we abusively write S instead of $(2 \cdot S)$. Assume $S = (2 \cdot T_{\sigma(1)}) \to \ldots \to (2 \cdot T_{\sigma(n)}) \to o$ and $T = (2 \cdot S_1) \to \ldots \to (2 \cdot S_n) \to o$. We first prove that the first point of the claim reduces to the second one.

Since the context in $\vdash t : (2 \cdot S) \to T$ is empty, the head variable of t is bound and the arity of t is $\geqslant 1$. Thus, $t = \lambda x_0.\lambda x_1 \ldots x_p.x\, t_1 \ldots t_q$ with $t_1, \ldots, t_q$ normal forms whose respective head variables are denoted $y_1, \ldots, y_q$. Note that:

- x is $x_1, \ldots, x_p$ or x_0 since x is bound.
- The type assigned to x_0 is S. The respective types assigned to $x_1, \ldots x_p$ are $S_1, \ldots, S_p$.
- The common target type of T and the type of x_0 is o.

Since (S,T) is proper, o does not occur in $S_1, \ldots, S_n$, so necessarily, $x = x_0$ and $x : (2 \cdot S) \vdash \lambda x_1 \ldots x_p.x\, t_1 \ldots t_q : T$ is derivable by means of a quantitative derivation P_*. Thus, we are now in the second case. The type of $x\, t_1 \ldots t_q$ is both $T_{\sigma(q+1)} \to \ldots \to T_{\sigma(n)} \to o$ since $x : S$ and $S_{p+1} \to \ldots \to S_n \to o$ since $t : T$, so $p = q$ and for $p+1 \leqslant i \leqslant n$, $\sigma(i) = i$ and $S_i = T_i$. Let us denote $o_1, \ldots, o_n$ the respective target types of $S_1, \ldots, S_n$. Since the type of x is S, the respective types of $t_1, \ldots, t_p$ must be $T_{\sigma(1)}, \ldots, T_{\sigma(p)}$. Moreover, since the "tail" of T is made of singleton sequence types $(2 \cdot S_i)$, $t_1, \ldots, t_p$ are typed once in P and the head variables $y_1, \ldots, y_p$ of $t_1, \ldots t_p$ are also typed exactly once. In particular, P_* has a subderivation at depth n of the form:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : (2 \cdot S) \vdash x : S}\,\mathtt{ax} \qquad P_1 \rhd C_1 \vdash t_1 : T_{\sigma(1)}\ [2]}{x : (2 \cdot S), C_1 \vdash x\, t_1 : (2 \cdot S_{\sigma(2)}) \to \ldots \to o}\,\mathtt{app} \qquad \ldots}{\ldots}\,\mathtt{app} \qquad P_p \rhd C_p \vdash t_p : T_{\sigma(p)}\ [2]}{x : (2 \cdot S), C_1 \uplus \ldots \uplus C_p \vdash x\, t_1 \ldots t_p : T'}\,\mathtt{app}$$

where $T' = T_{\sigma(p+1)} \to \ldots \to T_{\sigma(n)} \to o = S_{p+1} \to \ldots \to S_n \to o$.

Let us prove now that, for all $1 \leqslant i \leqslant p$, the unique argument derivation of x in P typing t_i, that we denote P_i, concludes with $x_{\sigma(i)} : (2 \cdot S_{\sigma(i)}) \vdash t_i : T_{\sigma(i)}$.

First, since t_i is normal, $t_i = \lambda z_1 \ldots z_{p'}.y_i\, u_1 \ldots u'_q$. Since $t_i : T_{\sigma(i)}$, t_i is free-headed by the observation above. Moreover, the head variable of t_i is typed once in P_* since t_i is typed once. Thus, y_i is one of the $x_1, \ldots, x_p$. The only possibility is $y_i = x_{\sigma(i)}$ since the types of $x_1, \ldots, x_p$ have pairwise distinct targets.

Since P is quantitative and $(2 \cdot S_i)$ is a *singleton* sequence type, $x_1, \ldots,\ x_p$ must be exactly typed once in P_*, the subderivation of P typing $x\, t_1 \ldots t_p$. This entails that the $\mathtt{ax}$-rule typing $x_{\sigma(i)}$ as the head variable of t_i concludes with $x_{\sigma(i)} : (2 \cdot S_{\sigma(i)}) \vdash x_{\sigma(i)} : S_{\sigma(i)}$. Thus, P_i concludes with a judgment of the form $x_{\sigma(i)} : (2 \cdot S_{\sigma(i)}) \uplus C'_i \vdash t_i : T_{\sigma(i)}$ (2nd argument).

Since $\uplus_{1 \leqslant i \leqslant p}(x_{\sigma(i)} : (2 \cdot S_{\sigma(i)}) \uplus C'_i) = x_1 : (2 \cdot S_1), \ldots, x_p : (2 \cdot S_p)$, we deduce that C'_i is empty for all $1 \leqslant i \leqslant p$ (3rd argument). Thus, P_i concludes with $x_{\sigma(i)} : (2 \cdot S_{\sigma(i)}) \vdash t_i : T_{\sigma(i)}$.

This easily implies that $x_1 : (2 \cdot S_1) \vdash t_{\sigma^{-1}(1)} : T_1, \ldots x_p : (2 \cdot S_p) \vdash t_{\sigma^{-1}(p)} : T_p$ are judgments of P_*. In particular, they are approximably derivable. Thus, $t_1, \ldots, t_p$ also satisfy the hypothesis of point 2 of the claim. Since $t = \lambda x_1 \ldots x_p.x\, t_1 \ldots t_p$, we conclude using Definition 1. ◁

The two claims, which are valid for 001-normal forms, along with infinitary subject reduction and expansion, give a type-theoretical characterization of hereditary permutators in system $\mathtt{S}$:

▶ **Theorem 22.** *Let $t \in \Lambda^{001}$. Then t is a hereditary permutator iff $\vdash t : (2 \cdot S) \to T$ is approximably derivable for some proper permutator pair (S,T).*

Proof.

- The implication $\Leftarrow$ is given by Claim 21 and Proposition 17.
- Implication $\Rightarrow$: let t be a hereditary permutator. By Definition 1, its Böhm tree is of the form $\lambda x.h$ where h is a normal x-headed hereditary permutator. By Claim 20, there is a proper permutator pair (S,T) and an approximable derivation P such that $P \rhd x : (2 \cdot S) \vdash h : T$. By Proposition 17, $\vdash t : (2 \cdot S) \to T$ is also approximably derivable. ◀

3 A unique type to rule them all

In this section, we explain how to enrich system $\mathtt{S}$ with type constants and typing rules so that there is one type characterizing the set of hereditary permutators, as expected.

In Section 2, we proved that a term t is a hereditary permutator iff it can be assigned a type of the form $(2{\cdot}S) \to T$ where (S,T) is a proper permutator pair. To obtain a unique type for all the hereditary permutators, one idea is to identify all the types of the form $(2{\cdot}S) \to T$, where (S,T) ranges over $\mathtt{PPP}$ with a type constant $\mathtt{ptyp}$. However, since quotienting types may bring unsoundness (e.g., if o and $o \to o$ are identified), one must then verify that the correctness and the completeness of system $\mathtt{S}$ is preserved, and that the approximability criterion can be suitably extended. The main argument, given by Lemma 26, is that the notions of hereditary permutators and permutators pairs, which are infinitary, have arbitrarily big finite approximations, which are defined as trucations at some applicative depth d. Thus, we may express hereditary permutators and permutator pairs as asymptotic limits and adapt the general methods of system $\mathtt{S}$.

Our approach parallels that of Tatsuta [14], which uses a family of constants $\mathtt{ptyp}_d$, with a few differences: in the finite restriction of our system, it is easier to deal with hereditary permutators (normalization is simple to prove in finite non-idempotent type systems), but of course we have to treat the infinitary typings and we consider the constant $\mathtt{ptyp}$, which is subsumed under all the $\mathtt{ptyp}_d$, which represent hereditary permutators under level d.

3.1 Permutator schemes

Before presenting the system giving a unique type to all hereditary permutators, we must first explain how the typings of hereditary permutators are approximated in system $\mathtt{S}$.

▶ **Definition 23** (Permutator schemes). *Let $d \geqslant 0$. A term t is a x-headed (resp. closed) permutator scheme of degree d if its Böhm tree is equal to that of a hereditary permutator on $\{b \in \{0,1,2\}^* \mid \mathtt{ad}(b) \leqslant d\}$. The set of x-headed (resp. closed) permutators schemes of degree d is denoted $\mathtt{PS}_d(x)$ (resp. $\mathtt{PS}_d$).*

The sequence $(\mathtt{PS}_d)$ is decreasing, i.e., $\mathtt{PS}_d \supseteq \mathtt{PS}_{d+1}$, and $\mathtt{HP} = \cap_{d \geqslant 0} \mathtt{PS}_d$.

▶ **Definition 24** (Permutator pairs of degree d). *Let $d \in \mathbb{N}$.*

- *When o ranges over $\mathcal{O}$, the set $\mathtt{PP}_d(o)$ of **o-permutator pairs of degree d** (S,T), where S and T are $\mathtt{S}$-types, is defined by induction on d:*

$$\frac{}{(\underbrace{() \to \ldots ()}_{n} \to o, \underbrace{() \to \ldots \to ()}_{n} \to o) \in \mathtt{PP}_0(o)}$$

$$\frac{(S_1,T_1) \in \mathtt{PP}_{d-1}(o_1), \ldots, \;\; (S_n,T_n) \in \mathtt{PP}_{d-1}(o_n) \quad o_1,\ldots,o_n,o \textit{ pairwise distinct} \quad \sigma \in \mathfrak{S}_n}{((2\cdot T_{\sigma(1)}) \to \ldots \to (2\cdot T_{\sigma(n)}) \to o, (2\cdot S_1) \to \ldots (2\cdot S_n) \to o) \in \mathtt{PP}_d(o)}$$

- *A pair $(S,T) \in \mathtt{PP}_d(o)$ is said to be **proper** if every type variable occurs at most once in S and T. The set of proper permutator pairs of degree d is denoted $\mathtt{PPP}_d$.*

We can also see permutator pairs of degree d as truncation of permutator pairs: let U be a $\mathtt{S}$-type or a sequence type and $d \in \mathbb{N}$. We denote by $(U)^{\leqslant d}$ the truncation of T at depth d i.e., $\mathtt{supp}((U)^{\leqslant d}) = \mathtt{supp}(U) \cap \{c \in \mathbb{N}^* \,|\, \mathtt{ad}(c) \leqslant d\}$ and $(U)^{\leqslant d}(c) = U(c)$ for all $c \in \mathtt{supp}((U)^{\leqslant d})$. It is easy to check that $(U)^{\leqslant d}$ is a correct type or sequence type. We extend the notation to $\mathtt{S}$-contexts. Note that, if $d \geqslant 1$, $((S_k)_{k\in K} \to T)^{\leqslant d} = ((S_k)^{\leqslant d-1})_{k \in K} \to (T)^{\leqslant d}$

and $d = 1$, then $((S_k)_{k\in K} \to T)^{\leqslant d} = () \to (T)^{\leqslant 1}$. By induction on d, this entails that, if $(S,T) \in \mathtt{PPP}$, then $((S)^{\leqslant d}, (T)^{\leqslant d}) \in \mathtt{PPP}_d$. Indeed, the base case $(d = 0)$ is obvious and if $d \geqslant 1$, $T = (2 \cdot S_1) \to \ldots \to (2 \cdot S_n) \to o$ and $S = (2 \cdot T_{\sigma(1)}) \to \ldots \to (2 \cdot T_{\sigma(n)}) \to o$ with $\sigma \in \mathfrak{S}_n$, $(S_i, T_i) \in \mathtt{PPP}$ for $1 \leqslant i \leqslant n$, then:

- $(T)^{\leqslant d} = (2 \cdot (S_1)^{\leqslant d-1}) \to \ldots \to (2 \cdot (S_1)^{\leqslant d-1}) \to o$ (eq$_3$)
- $(S)^{\leqslant d} = (2 \cdot (T_{\sigma(1)})^{\leqslant d-1}) \to \ldots \to (2 \cdot (T_{\sigma(n)})^{\leqslant d-1}) \to o$ (eq$_4$)

so that, by Definition 24, $((S)^{\leqslant d}, (T)^{\leqslant d}) \in \mathtt{PPP}_d(o)$.

▶ **Proposition 25** (Characterizing permutation schemes). *Let $d \geqslant 1$ and t be a 001-term. Then $t \in \mathtt{PS}_d$ iff $\vdash t : (2 \cdot S) \to T$ is approximably derivable for some $(S,T) \in \mathtt{PPP}_d$.*

Proof.

$\Rightarrow$ Straightforward induction on the structure of t.

$\Leftarrow$ The proof is the same as Claim 21, we also obtain that $x_i : (2 \cdot S_i) \vdash t_{\sigma^{-1}(i)} : T_i$ are judgments of P, except that $(S_i, T_i) \in \mathtt{PPP}_{d-1}$ instead of $(S_i, T_i) \in \mathtt{PPP}$. ◀

It is not enough to know that a x-headed hereditary permutator t is approximably typable in a judgment $x : (2 \cdot S) \vdash t : T$ with $(S,T) \in \mathtt{PPP}$, which implies that T is the supremum of a direct family of finite types which be assigned to t: in order to prove soundness regarding quotienting, we must prove that this typing is the supremum of typings ensuring that t is a permutator scheme of degree d, i.e., by Proposition 25, one must type t with $(S_d, T_d) \in \mathtt{PPP}_d$ for all d. The lemma below is the missing third ingredient (along with Claims 20 and 21) of this article and will allow us to define in Section 3.2 an extension of system $\mathtt{S}$ giving a unique type to hereditary permutators:

▶ **Lemma 26** (Approximations and permutator pairs). *If $P \rhd x : (2{\cdot}S) \vdash t : P$ is approximable, where $(S,T) \in \mathtt{PPP}$, then, for all $d \in \mathbb{N}$, there is a finite $P_d \leqslant P$ such that $P_d \rhd x : (2 \cdot S_d) \vdash t : T_d$ with $(S_d, T_d) \in \mathtt{PPP}_d$.*

Proof. Since $()$ does not occur in S and T, by Lemma 18, we can assume that t is a 001-normal form without loss of generality. We then reason by induction on d.

Let us present the argument informally. Say that $t = \lambda x_1 \ldots x_p.x\, t_1 \ldots t_p$, $S = T_{\sigma(1)} {\to} \ldots {\to} T_{\sigma(n)} {\to} o$ and $T = S_1 {\to} \ldots {\to} S_n {\to} o$. Intuitively, $t : T$ with $x : S$ and for $1 \leqslant i \leqslant p$, $t_i : T_{\sigma(i)}$ is a hereditary permutator headed by $x_{\sigma(i)} : S_{\sigma(i)}$, as specified by Fig. 1. When we truncate the type of t at applicative depth d, we have now $t : (T)^{\leqslant d}$ with $x : (S)^{\leqslant d}$. But, by (eq$_3$) and (eq$_4$), we must truncate the types of $t_1, \ldots, t_p$ and $x_1, \ldots, x_p$ at applicative depth $d-1$. Inductively, this demands that we truncate the types of the arguments of $t_1, \ldots, t_p$ at applicative depth $d-2$. By proceeding so, we obtain a finite derivation $P_d \leqslant P$ concluding with $x : (2 \cdot (S)^{\leqslant d}) \vdash t : (T)^{\leqslant d}$. ◀

3.2 System $\mathtt{S}_{\mathtt{hp}}$

Let $\mathtt{ptyp}$ and $\mathtt{ptyp}_d$ ($d \in \mathbb{N}$) be a family of type constants. The set of $\mathtt{S}_{\mathtt{hp}}$-types is defined by:

$$T,\, S_k \ ::= \ o \parallel \mathtt{ptyp}_d \parallel \mathtt{ptyp} \parallel (S_k)_{k\in K} \to T$$

System $\mathtt{S}_{\mathtt{hp}}$ has the same typing rules as system $\mathtt{S}$ with the addition of:

$$\frac{x : (2 \cdot S) \vdash t : T \qquad (S,T) \in \mathtt{PPP}_d}{\vdash \lambda x.t : \mathtt{ptyp}_d}\ \mathtt{hp}_d \qquad\qquad \frac{x : (2 \cdot S) \vdash t : T \qquad (S,T) \in \mathtt{PPP}}{\vdash \lambda x.t : \mathtt{ptyp}}\ \mathtt{hp}$$

Thus, rule $\mathtt{hp}_d$ allows assigning the constant $\mathtt{ptyp}_d$ to any normal permutator scheme of degree d and rule $\mathtt{hp}$ assign the constant $\mathtt{ptyp}$ to any normal hereditary permutator by Claims 20 and 21. Intuitively, $\mathtt{ptyp} = \mathtt{ptyp}_\infty$ and we will make this idea more precise with Definition 27. Note also that if $t : \mathtt{ptyp}_d$ or $t : \mathtt{ptyp}$, t cannot be applied to an argument u, even if t is an abstraction: the rules $\mathtt{hp}_d/\mathtt{hp}$ freeze the terms.

The notions of support, bisupport, permutator pairs *etc* naturally extend to $\mathtt{S}_{\mathtt{hp}}$. We define an order $\leqslant$ on $\mathcal{O} \cup \{\rightarrow, \mathtt{ptyp}\} \cup \{\mathtt{ptyp}_d \,|\, d \in \mathbb{N}\}$ by $o \leqslant o$, $\rightarrow \leqslant \rightarrow$, $\mathtt{ptyp}_d \leqslant \mathtt{ptyp}$ and $\mathtt{ptyp}_d \leqslant \mathtt{ptyp}_{d'}$ for $d \leqslant d'$.

▶ **Definition 27** (Approximation and Approximability in system $\mathtt{S}_\mathtt{h}$)**.**

- *Let P_0 and P be two $\mathtt{S}_{\mathtt{hp}}$-derivations. We write $P_0 \leqslant P$ (P_0 is an approximation of P) if $\mathtt{bisupp}(P_0) \subseteq \mathtt{bisupp}(P)$ and, for all $\mathtt{p} \in \mathtt{bisupp}(P_0)$, $P_0(\mathtt{p}) \leqslant P(\mathtt{p})$.*
- *Let P be a $\mathtt{S}_{\mathtt{hp}}$-derivation. Then P is* ***approximable*** *if P is the supremum of its finite approximations.*

This extends Definition 10: for all $\mathtt{S}$-derivations P, P is approximable for system $\mathtt{S}$ iff it is approximable for system $\mathtt{S}_{\mathtt{hp}}$. We first notice that rules $\mathtt{hp}_{(d)}$ are invertible for HNF:

▶ **Lemma 28** (Inverting rules ($\mathtt{hp}_d$) for head normal forms)**.** *Let t be a HNF. If $\vdash t : \mathtt{ptyp}_d$ (resp. $P \rhd \vdash t : \mathtt{ptyp}$) is approximably derivable, then $t = \lambda x.t_0$ with $x : (2 \cdot S) \vdash t_0 : T$ approximably derivable, for some $(S, T) \in \mathtt{PPP}_d$ (resp. $(S, T) \in \mathtt{PPP}$).*

Proof. We consider the case $\mathtt{ptyp}$ (the case $\mathtt{ptyp}_d$ is similar), i.e., we assume that $P' \rhd \vdash t : \mathtt{ptyp}$ is approximable. For one, $t = x\, t_1 \ldots x_n$ is impossible, because we would have $C(x) \neq (\,)$ since the head variable x is free in $x\, t_1 \ldots t_n$. So, t is an abstraction, i.e., $t = \lambda x.t_0$ and thus, the last rule of P is either $\mathtt{abs}$, $\mathtt{hp}_d$, $\mathtt{hp}$. But it is neither $\mathtt{abs}$ (we would have an arrow type) nor $\mathtt{hp}_d$, so it is $\mathtt{hp}$ and thus, P' is of the form:

$$P' = \cfrac{P \rhd x : (2 \cdot S) \vdash t_0 : T \qquad (S, T) \in \mathtt{PPP}}{\vdash t : \mathtt{ptyp}}\ \mathtt{hp}$$

Since P' is approximable, P also is. ◀

All is now in place to obtain the expected properties of system $\mathtt{S}_{\mathtt{hp}}$:

▶ **Lemma 29** (Characterizing normal hereditary permutators)**.** *Let t be a 001-normal form.*

- *$t \in \mathtt{PS}_d$ iff $\vdash t : \mathtt{ptyp}_d$ is approximably derivable.*
- *$t \in \mathtt{HP}$ iff $\vdash t : \mathtt{ptyp}$ is approximably derivable.*

Proof. The two points are handled similarly. We do not prove the first one, which uses Proposition 25:

- If $t = \lambda x.h$ is a HP, then, by Claim 20, there is $(S, T) \in \mathtt{PPP}$ and P a $\mathtt{S}$-derivation such that $P \rhd x : (2 \cdot S) \vdash h : T$. We then set:

  $$P' = \cfrac{P \rhd h : (2 \cdot S) \vdash p : T}{\vdash t : \mathtt{ptyp}}\ \mathtt{hp}$$

  By Lemma 26, for all $d \in \mathbb{N}$, there is a finite $\mathtt{S}$-derivation $P_d \leqslant P$ such that $P_d \rhd x : (2 \cdot S_d) \vdash h : T_d$ with $(S_d, T_d) \in \mathtt{PPP}_d$ and $P = \sup_d P_d$. We then set:

  $$P'_d = \cfrac{P_d \rhd x : (2 \cdot S_d) \vdash h : T_d}{\vdash t : \mathtt{ptyp}_d}\ \mathtt{hp}_d$$

  By construction, $\sup_d P'_d = P'$.

- Conversely, assume that $P' \rhd \vdash t : \texttt{ptyp}$ is approximable. By Lemma 28, P' concludes with the $\texttt{hp}$-rule, so P' is of the form:

$$P' = \cfrac{P \rhd x : (2 \cdot S) \vdash t_0 : T \qquad (S, T) \in \texttt{PPP}}{\vdash t : \texttt{ptyp}}\ \texttt{hp}$$

  Let $d \in \mathbb{N}$. Since P' is the supremum of its finite approximations, there is a finite approximation $P'_0 \leqslant P'$ concluding with[1] $\vdash t : \texttt{ptyp}$ or $\vdash t : \texttt{ptyp}_{d'}$ with $d' \geqslant d$. Thus, $t \in \texttt{PS}_{d'} \subseteq \texttt{PS}_d$ or $t \in \texttt{HP}$. This proves that $t \in \cap_{d \geqslant 0} \texttt{PS}_d = \texttt{HP}$. ◀

▶ **Lemma 30** (Soundness of system $\texttt{S}_{\texttt{hp}}$). *If t is approximably typable in system $\texttt{S}_{\texttt{hp}}$, then t is head normalizing.*

Proof. If t is approximably typable, there is a *finite* $\texttt{S}_{\texttt{hp}}$-derivation $P \rhd C \vdash t : T$. If t is a HNF, we are done. In the other case, $t \to_{\texttt{h}} t'$ for some t'. It is routine work in *non-idempotent* intersection type theory (see [4]) to prove a *weighted* subject reduction property, i.e., that there is $P' \rhd C \vdash t' : T$ such that $|\texttt{supp}(P')| < |\texttt{supp}(P)|$, i.e., P' contains strictly less judgments than P does. The only unusual rules are $\texttt{hp}_d$ and $\texttt{hp}$, which are easily handled.

Since $|\texttt{supp}(P)| \in \mathbb{N}$ and $\mathbb{N}$ is well-founded, weighted subject reduction entails that head reduction terminates on t. ◀

▶ **Corollary 31.** *If $\vdash t : \texttt{ptyp}$ is approximably derivable, then t is WN_∞.*

Proof. By Lemma 30, t reduces to a HNF t'. By subject reduction, $\vdash t' : \texttt{ptyp}$ is also approximably derivable. Then, Lemma 28 entails that $t = \lambda x.t_0$ and $x : (2 \cdot S) \vdash t_0 : T$ is approximably derivable in system $\texttt{S}$ for some t_0 and permutation pair (S, T). Since this latter judgment is ( )-free, Proposition 15 entails that t_0 is WN_∞. Thus, t also is WN_∞. ◀

More generally, the dynamic properties of system $\texttt{S}$ are preserved in system $\texttt{S}_{\texttt{hp}}$.

▶ **Proposition 32** (Infinitary subject reduction). *If $t \to_\beta^\infty t'$ and $P \rhd C \vdash t : T$ is an approximable $\texttt{S}_{\texttt{hp}}$-derivation, then there exists an approximable derivation $P' \rhd C \vdash t' : T$.*

▶ **Proposition 33** (Infinitary subject expansion). *If $t \to_\beta^\infty t'$ and $P' \rhd C \vdash t' : T$ is an approximable $\texttt{S}_{\texttt{hp}}$-derivation, then there exists an approximable derivation $P \rhd C \vdash t : T$.*

Proof. The proofs of infinitary subject reduction and expansion in system $\texttt{S}_{\texttt{hp}}$ do not differ of those for system $\texttt{S}$, which can be found in [17] (in particular, Section II.D. and VI.D.) or in Chapter 10 of [18], so we do not give the details. Once again, the only new rules are $\texttt{hp}$ and $\texttt{hp}_d$, which are easily handled in the one step case, then in the asymptotic case.

Infinitary subject reduction is easy to prove, but infinitary subject expansion holds because we can expand *finite* approximations of a derivation P' concluding a productive reduction path. Why? Because, if $t \to_\beta^\infty t'$ and P'_f is finite and types t', then there is a term $t \to_\beta^k t_k$ obtained from t after a finite number k of β-reduction steps such that, on $\texttt{supp}(P')$, t_k and t' induce the same subtree (this is a consequence of Definition 5). Thus, we can *substitute* t' with t_k in P'_f: we then obtain P_f^k typing t_k. After k expansion steps, we obtain from P_f^k a derivation P_f typing t. To conclude this dense summary, let us just say that the infinitary subject expansion is not so about the rules than about the possibility to replace a derivation by its finite approximations, which is precisely what Definition 27 captures for system $\texttt{S}_{\texttt{hp}}$. ◀

---

[1] The case $P'_0 \rhd \vdash t : \texttt{ptyp}$ is possible: there are finite HP and PPP, e.g., $\lambda x.x$ and (o, o).

We now give a positive answer to TLCA Problem # 20:

▶ **Theorem 34** (Characterizing hereditary permutators with a unique type). *Let* $t \in \Lambda^{001}$. *Then* t *is a hereditary permutator iff* $\vdash t : \mathtt{ptyp}$ *is approximably derivable in system* $\mathtt{S}_{\mathtt{hp}}$.

Proof.

$\Rightarrow$ If t is a HP, let t' be its 001-NF. By Lemma 29, there is an approximable derivation $P' \rhd \vdash t' : \mathtt{ptyp}$. By Proposition 33, there is $P \rhd \vdash t' : \mathtt{ptyp}$ approximable.

$\Leftarrow$ Given by Corollary 31. ◀

Future work

We plan to adapt system $\mathtt{S}$ to characterize other sets of Böhm trees and other notions of infinitary normalization, including weak normalization in the calculi Λ^{101} and Λ^{111} of [11].

References

1. Patrick Bahr. Strict Ideal Completions of the Lambda Calculus. In *FSCD 2018, July 9-12, Oxford*, pages 8:1–8:16, 2018.
2. Henk Barendregt. *The Lambda-Calculus: Its Syntax and Sematics*. Ellis Horwood series in computers and their applications. Elsevier, 1985.
3. Jan A. Bergstra and Jan Willem Klop. Invertible Terms in the Lambda Calculus. *Theor. Comput. Sci.*, 11:19–37, 1980.
4. Antonio Bucciarelli, Delia Kesner, and Daniel Ventura. Non-Idempotent Intersection Types for the Lambda-Calculus. *Mathematical Structures in Computer Science.*, 2017.
5. Daniel De Carvalho. *Sémantique de la logique linéaire et temps de calcul*. PhD thesis, Université Aix-Marseille, November 2007.
6. Mario Coppo and Mariangiola Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. *Notre Dame Journal of Formal Logic*, 4:685–693, 1980.
7. Haskell B. Curry and Robert Feys. *Combinatory Logic*, volume I. North-Holland Co., Amsterdam, 1958. (3rd edn. 1974).
8. Lukasz Czajka. A Coinductive Confluence Proof for Infinitary Lambda-Calculus. In *Rewriting and Typed Lambda Calculi - Joint International Conference, RTA-TLCA, Vienna, Austria, July 14-17*, pages 164–178, 2014.
9. Mariangiola Dezani-Ciancaglini. Characterization of Normal Forms Possessing Inverse in the *lambda-beta-eta*-Calculus. *Theor. Comput. Sci.*, 2(3):323–337, 1976.
10. Philippa Gardner. Discovering Needed Reductions Using Type Theory. In *TACS, Sendai*, 1994.
11. Richard Kennaway, Jan Willem Klop, M. Ronan Sleep, and Fer-Jan de Vries. Infinitary Lambda Calculus. *Theor. Comput. Sci.*, 175(1):93–125, 1997.
12. Betti Venneri Mario Coppo, Mariangiola Dezani-Ciancaglini. Functional Characters of Solvable Terms. *Mathematical Logic Quarterly*, 27:45–58, 1981.
13. Makoto Tatsuta. Types for Hereditary Head Normalizing Terms. In *FLOPS, Ise, Japan, April 14-16*, pages 195–209, 2008.
14. Makoto Tatsuta. Types for Hereditary Permutators. In *LICS, 24-27 June, Pittsburgh*, pages 83–92, 2008.
15. Steffen van Bakel. Complete Restrictions of the Intersection Type Discipline. *Theor. Comput. Sci.*, 102(1):135–163, 1992.
16. Steffen van Bakel. Intersection Type Assignment Systems. *Theor. Comput. Sci.*, 151(2):385–435, 1995.
17. Pierre Vial. Infinitary intersection types as sequences: a new answer to Klop's problem. In *LICS, Reykjavik*, 2017.
18. Pierre Vial. *Non-Idempotent Typing Operator, beyond the Lambda-Calculus*. Phd thesis, Université Sorbonne Paris-Cité, 2017, available on `http://www.irif.fr/~pvial`.

Model Checking Strategy-Controlled Rewriting Systems

Rubén Rubio
Facultad de Informática, Universidad Complutense de Madrid, Spain
rubenrub@ucm.es

Narciso Martí-Oliet
Facultad de Informática, Universidad Complutense de Madrid, Spain
narciso@ucm.es

Isabel Pita
Facultad de Informática, Universidad Complutense de Madrid, Spain
ipandreu@ucm.es

Alberto Verdejo
Facultad de Informática, Universidad Complutense de Madrid, Spain
jalberto@ucm.es

Abstract

Strategies are widespread in Computer Science. In the domain of reduction and rewriting systems, strategies are studied as recipes to restrict and control reduction steps and rule applications, which are intimately local, in a derivation-global sense. This idea has been exploited by various tools and rewriting-based specification languages, where strategies are an additional specification layer. Systems so described need to be analyzed too. This article discusses model checking of systems controlled by strategies and presents a working strategy-aware LTL model checker for the Maude specification language, based on rewriting logic, and its strategy language.

2012 ACM Subject Classification Software and its engineering → Model checking; Theory of computation → Equational logic and rewriting; Theory of computation → Rewrite systems

Keywords and phrases Model checking, strategies, Maude, rewriting logic

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.34

Category System Description

Related Version An extended version of the paper is available at `http://maude.sip.ucm.es/strategies/tr0219.pdf` ([30]).

Funding Research partially supported by MCIU Spanish project TRACES (TIN2015-67522-C3-3-R).
Rubén Rubio: Partially supported by MCIU grant FPU17/02319.

Acknowledgements We are grateful to the reviewers for their careful reading, and their detailed and very useful comments.

1 Introduction

Strategies are meaningful for artificial intelligence, games, semantics of programming languages, automated reasoning, etc. Although their purposes and formalizations differ, they all honor the Greek etymology of the word, meaning *the office of a general*, who is in charge of the overall planning of the operations. In the modeling of concurrent systems using rewriting techniques, strategies are a useful resource to capture the global behavior of the intended models. Since rewriting consists of a successive, non-deterministic and somehow unrelated application of rules anywhere within a term, strategies have been studied in deep [32], and

© Rubén Rubio, Narciso Martí-Oliet, Isabel Pita, and Alberto Verdejo;
licensed under Creative Commons License CC-BY
4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019).
Editor: Herman Geuvers; Article No. 34; pp. 34:1–34:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

different definitions have been proposed [8, 19]. Strategies as a first-class object have been exploited in tools like Stratego [10], Tom [4], and the specification languages ELAN [7], and Maude [13].

Model checking [11] is a consolidated formal method, which still evolves in multiple directions. Its classical setting is transition systems, where the notion of strategy is naturally defined. This paper studies the satisfaction of temporal properties by models controlled by strategies and how it can be checked, and applies the method to the Maude strategy language by implementing a strategy-aware model checker. It is built as an extension of the existing Maude LTL model checker [16] for systems specified in rewriting logic, already applied to various interesting systems [6, 23, 28].

Strategies and model checking together have already been addressed in the literature, but with a different approach and objectives. In the context of multiplayer games, several logics have been proposed to *reason* about player strategies like ATL* [1] and *strategy logic* [27]. However, strategies are not provided as input but quantified in the formula, and they are not represented explicitly. Other logics take past actions into account to condition its requirements like mCTL* [21]. In our case, strategies are part of the model specification while the property logic remains unaltered. As well, strategies should not be confused with heuristics to guide the search in the model-checker algorithms [14].

After reviewing the model-checking framework and strategies as defined in the literature, this paper discusses model checking linear temporal properties on strategy-controlled systems and a model transformation is proposed to match the classical setting and allow using the standard algorithms. Following a short introduction to rewriting logic [26], Maude [12], and its strategy language [15], we propose a small-step operational semantics from which model checking is defined according to the previous approach. The strategy-aware LTL model checker we have implemented is then described and illustrated by an example. The extended version of this document [30], the complete semantics of the Maude strategy language, and the model checker itself are all available at `http://maude.sip.ucm.es/strategies`.

2 Preliminaries

2.1 Model checking

Model checking [11] is a well-established formal method to ensure or refute the correctness of a model according to a temporal specification. In the classical setting, models are based on transition systems, formally described by Kripke structures [20] $\mathcal{K} = (S, \to, I, AP, \ell)$ where

- S is the set of states,
- $(\to) \subseteq S \times S$ is a serial binary relation on S,
- $I \subseteq S$ is a finite set of initial states,
- AP is a finite set of *atomic propositions*, and
- $\ell : S \to \mathcal{P}(AP)$ labels each state with the propositions that hold on it.

In turn, the property is expressed by a formula φ in some temporal logic like CTL, CTL* or LTL, which describes the intended behavior in terms of the atomic properties $p, q, \ldots \in AP$ combined by different temporal operators. The model-checking problem, deciding whether the model satisfies the formula $\mathcal{K} \vDash \varphi$, is decidable for any of the previous logics, a decidable transition relation, and a finite S. However, for both CTL* and LTL, model checking is PSPACE-complete and the models of interest usually have a huge number of states. In various situations, the expectation of refuting correctness in reasonable time is good enough.

The actual model executions $\pi = (s_k)_{k=1}^\infty$ leave propositional traces $\ell(\pi) := (\ell(s_k))_{k=1}^\infty$, from which the satisfaction of the formula is decided.

$$\begin{array}{lccccccl} \pi & s_1 \longrightarrow s_2 \longrightarrow \cdots \longrightarrow s_n \longrightarrow \cdots & \in S^\omega \\ \ell(\pi) & \{p\} \quad \emptyset \quad \cdots \quad \{p,q\} \quad \cdots & \in \mathcal{P}(AP)^\omega \end{array}$$

Linear-time properties can always be characterized by a satisfaction relation $\ell(\pi) \vDash \varphi$ on propositional traces and an implicit universal quantification over all model paths π. Differently, branching-time properties combine universal and existential requirements on any state of the derivation, so that the execution should be seen as a tree.

2.2 Strategies

In rewriting systems, rules typically represent local transitions that are often not enough to describe complex computations. These intricacies are usually expressed at a higher level, describing how rules should be applied. This is the task of strategies, whose study goes back to the λ-calculus, as fixed criteria for selecting the next redex to be reduced [5]. Later, strategies were allowed to be aware of the derivation history and to be *explicit* [2, §11.5], expressed as programs that control the application of rules. We are interested in the latter kind of strategies, for which different abstract descriptions and practical representations have been proposed and implemented [8, 19].

Strategies are properly defined in the context of *abstract reduction systems* (ARS). An ARS [2] $\mathcal{A} = (S, \to)$ is a set of states S endowed with a binary relation $\to$. An element $(s, s') \in (\to)$ or $s \to s'$ is called a *reduction step*, and a finite or infinite sequence of connected reduction steps $s_0 \to s_1 \to \cdots \to s_n$ is a *derivation*. We denote by $\Gamma_\mathcal{A}^\omega$ the set of infinite derivations of $\mathcal{A}$ seen as words in S^ω,

$$\Gamma_\mathcal{A}^\omega := \{s_0 s_1 \cdots s_n \cdots : s_k \in S \text{ and } s_k \to s_{k+1}, k \geq 0\},$$

$\Gamma_\mathcal{A}^*$ is the set of finite derivations as words in S^*, and $\Gamma_\mathcal{A} := \Gamma_\mathcal{A}^\omega \cup \Gamma_\mathcal{A}^*$ the union of both in $S^\infty := S^\omega \cup S^*$. Considering both finite and infinite derivations is tedious, but we are interested in modeling computations and proofs as well as reactive systems behavior, for which they are respectively relevant. We say that $\mathcal{A}$ is *finite* if S is finite, and $\mathcal{A}$ is *finitary* if for any $s \in S$ the states s' such that $s \to s'$ are finitely many.

Several definitions of strategies are reviewed in [8], but two general formalizations are specially discussed:

1. *Abstract* or *extensional strategies* are subsets of derivations $E \subseteq \Gamma_\mathcal{A}$, that is, languages in S^∞ whose words w are $\mathcal{A}$-derivations with $w_k \to w_{k+1}$.
2. *Intensional strategies* are defined as partial functions $\lambda : S^* \to \mathcal{P}(S \cup \{\top\})$ that decide the possible next steps according to the past of the derivation. They must satisfy that for all $s, s' \in S$ and $v \in S^*$, $s' \in \lambda(vs)$ must imply $s \to s'$. The symbol $\top$ indicates that the derivation can stop there. In case $\lambda(w) = \emptyset$, the derivation cannot stop or continue, so it is discarded. Hence, these strategies can attempt rewriting paths that may eventually fail.

Extensional strategies represent an abstract selection of ARS executions as a whole, while the more constructive intensional strategies determine the next reduction in each step. Unlike in [8], we have considered unlabeled transition systems to simplify the presentation. Since classical model checking only considers properties on the states, labels can be easily added without repercussions. Moreover, the definition of intensional strategies has been modified to include the $\top$ symbol. Otherwise, derivations could stop at any step, which is inconvenient in practice. These definitions fall into the class of *history aware strategies* of [32], and intensional strategies are similar to *non-deterministic strategies* in games, except that these may select the next player action instead of the next state.

▶ **Example 1.** Consider the ARS $(\{a,b\},\{(a,a),(a,b),(b,a)\})$

A strategy allowing at most one stay in b is described extensionally as $\{a\}^*\{b,\varepsilon\}(\{a\}^* \cup \{a\}^\omega)$, and intensionally as $\lambda(v) = \{a, \top\}$ if v contains a b, and $\lambda(v) = \{a, b, \top\}$ otherwise.

An intensional strategy induces an extensional one

$$E(\lambda) := \{w \in \Gamma_{\mathcal{A}} : w_i \in \lambda(w_0 \cdots w_{i-1}) \wedge (w \in S^\omega \vee \top \in \lambda(w))\}.$$

But the converse is not true, intensional strategies are less expressive than extensional ones. In the ARS of Example 1, the intensional strategy for $\{a\}^*$ has to be defined by $\lambda(v) = \{a, \top\}$ for any $v \in \{a\}^*$, since another a can always be added. And thus, the word a^ω would be included in $E(\lambda)$ by definition, so that $E(\lambda) \neq \{a\}^*$. Intuitively, intensional strategies decide on-the-fly while constructing the derivation, so they cannot decide on properties on the *infinity*. Languages recognized by automata with non-trivial acceptance conditions are examples of strategies that are necessarily extensional, but the more realistic devices or computations we are interested in modeling are very likely to be intensional. In any case, the extensional definition is simpler and will be useful.

Formally, intensional strategies are characterized as closed sets [8], which contain all words $w \in S^\omega$ whose finite prefixes are all prefixes of derivations within the strategy[1]. When discussing model checking, we will find an alternative characterization.

3 Strategy-aware model checking

Given a Kripke structure $\mathcal{K} = (S, \to, I, AP, \ell)$ and a strategy $E \subseteq \Gamma_{\mathcal{K}} := \Gamma_{(S,\to)} \cap IS^\infty$ starting from the initial states of $\mathcal{K}$, we want to give sense to model checking against a linear temporal formula φ and define the satisfaction relation $(\mathcal{K}, E) \models \varphi$. First, since temporal formulas are properly defined on infinite executions, we assimilate finite traces to infinite ones by repeating its last state forever. This is standard and fits with the idea of a finite machine that remains in its final state once stopped.

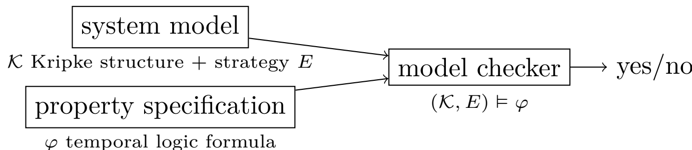


Figure 1 Model-checking procedure sketch.

Model checking a system controlled by a strategy against a linear-time property has an unavoidable and clear definition. As pointed in Section 2, the satisfaction of a linear property follows from a satisfaction relation $\rho \vDash \varphi$ on propositional traces $\rho \in \mathcal{P}(AP)^\omega$. Then,

▶ **Definition 2.** *Let φ be a linear formula, $(\mathcal{K}, E) \models \varphi$ if $\ell(\pi) \models \varphi$ for all $\pi \in E$.*

The set of allowed propositional traces $P = \{\, \rho \in \mathcal{P}(AP)^\omega : \rho \models \varphi \,\}$ completely defines the property, and the model-checking problem is the language containment problem $\ell(E) \subseteq P$, which is decidable and PSPACE-complete as long as $\ell(E)$ and P are ω-regular, like in the

[1] In fact, X^ω can be given a *prefix* topology (and even a distance) such that *closed* can be understood in the topological sense and the points so described are limits.

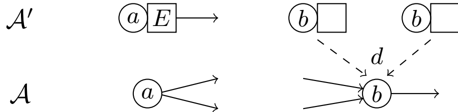


Figure 2 Strategy representation in an ARS (Definition 3).

LTL case [29]. Although this is a clear starting point and it would provide an actual algorithm if strategies were given as Büchi automata, this is not usually the case since they are rather expressed in some intensional or syntactical form like the Maude strategy language described in Section 4.1. Thus, to effectively decide model checking with standard algorithms, we propose to encode the model controlled by strategies as another abstract reduction system. Any intensional strategy can be so encoded, but the actual procedure will depend on the strategy representation. In Section 4.2, we do it for the Maude strategy language by means of a small-step operational semantics.

▶ **Definition 3.** *Given an ARS $\mathcal{A} = (S, \to)$ and a strategy $E \subseteq \Gamma_\mathcal{A}$, the pair $(\mathcal{A}, E)$ is represented in the ARS $\mathcal{A}' = (X, R)$ with descent function $d : X \to S$ if*

$$d(\Gamma_{\mathcal{A}'} \cap J\, X^\infty \cap (X^\omega \cup X^* F)) = E$$

for some sets $J, F \subseteq X$ of initial and final states of the strategy.

The representation $\mathcal{A}'$ simulates the system $\mathcal{A}$ constrained by E, in the sense that the executions of $\mathcal{A}'$ are the traces selected by the strategy. Intuitively, the representation states include something else to guarantee that the strategy is respected, which can be stripped with the descent function d. The initial states for the strategy execution are the subset J, since other states in X may represent ongoing strategy executions. The final set F is only required to distinguish admitted finite traces from incomplete ones. Still, we do not want them for model checking: the trace extension of the first paragraph can be implemented in the above representation by adding self-loops in F. However, this cannot be done without care. For example, observe the following encoding for the strategy $\{a, ab\}$, where a and b are final.

The extended language is $\{a^\omega, ab^\omega\}$ according to our criterion. However, a loop in a will allow spurious derivations like aab^ω. This can be solved by adding an extra copy of the final states, like in the right figure. Now, we assume that the strategies are over infinite words and define the concept of model checking.

▶ **Proposition 4.** *Let $\mathcal{K} = (S, \to, I, AP, \ell)$ be a Kripke structure, $E \subseteq \Gamma_\mathcal{K}$ a strategy, and (X, R) a representation of $(\mathcal{A}, E)$ with descent function d and initial states J, $(\mathcal{K}, E) \models \varphi$ iff $\mathcal{K}' \models \varphi$ where $\mathcal{K}' = (X, R, J, AP, \ell \circ d)$.*

That is, model checking $\mathcal{K}$ controlled by E is model checking its representation $\mathcal{K}'$. The reason is that the propositional traces of any such $\mathcal{K}'$ are exactly those of E. This method can be applied to any intensional strategy, but model checking will only be decidable if the ARS representation is finite.

▶ **Proposition 5.** *A strategy $E \subseteq \Gamma_\mathcal{A}$ on a finitary ARS $\mathcal{A}$ can be represented in a finitary ARS iff E is intensional, i.e. there is an intensional strategy λ such that $E = E(\lambda)$. In that case, it can be represented in a finite ARS iff E is ω-regular.*

We can draw from this proposition that a strategy on a finite ARS must be intensional and ω-regular to match the classical model-checking framework and apply its algorithms, because otherwise it cannot be represented in a finite Kripke structure.

Notice that the discussion in this section is restricted to linear-time properties. The representation of Definition 3 is not adequate for branching-time properties, since branches need not be preserved while descending with d. A suitable model-checking definition for logics like CTL* passes by model checking a more restricted representation, any bisimulation of the ARS (S^+, R) where $v\,R\,vs$ if $v \in \lambda(v)$ and λ is an intensional strategy.

4 Maude and its strategy language

Maude is a specification language [13, 12] based on *rewriting logic* [26], a general framework for modeling concurrency proposed in 1992 by José Meseguer. Its specifications are organized in modules of different kinds:

1. *Functional* modules define *membership equational logic* theories, composed of an order-sorted signature Σ, equations E, and sort membership axioms to express that a term t belongs to a sort s. Equations and membership axioms can be conditional.

$$(\forall X) \qquad \begin{array}{c} t = t' \\ t : s \end{array} \quad \text{if} \quad \bigwedge_i u_i = u_i' \;\wedge\; \bigwedge_j v_j : s_j$$

   For the specification to be executable, equations are oriented and functional modules must be confluent and terminating [12, §4.6]. Bidirectional relations, like commutativity, associativity and identity, are specified apart as *structural axioms*.

2. *System* modules specify rewriting theories $\mathcal{R} = (\Sigma, E, R)$ by adding rewriting rules R to a functional specification. Unlike equations, rewriting rules need be neither confluent nor terminating, so they are likely to express non-deterministic behavior.

$$(\forall X) \qquad t \Rightarrow t' \quad \text{if} \quad \bigwedge_i u_i = u_i' \;\wedge\; \bigwedge_j v_j : s_j \;\wedge\; \bigwedge_k w_k \Rightarrow w_k'$$

   However, rules are required to be coherent with equations and axioms [12, 26]. Conditional rules take a third type of conditions called rewriting conditions, that are satisfied if the term w_k can be rewritten to match w_k'.

The language syntax is a natural ASCII encoding of the mathematical notation above. Modules are a collection of declarations between `mod NAME is ... endm` (or `fmod`/`endfm` for functional modules). An operator $f : s_1 \times \cdots \times s_n \to s$ is defined as `op f : s1 .. sn -> s .` and structural axioms are inserted as attributes (`comm`, `assoc`, ...) between brackets. Equations are introduced by the keyword `eq` and rules by `rl`, prefixed by `c` if conditional.

▶ **Example 6.** The classical problem of the dining philosophers [18] is specified in the module `PHILOSOPHER-DINNER` of Listing 1. A philosopher is represented as a triple describing both hands contents (a fork ψ or nothing `o`) and an identifier. Rules `left` and `right` allow them to take the forks at their sides if they are in the table. The `release` rule restores both forks to the table. Since a circular table is represented by a list, we adopt the convention that the fork between the last and first philosophers is on the right, ensure it by the equation, and add a second `left` rule to allow the first philosopher to take this fork.

Listing 1 Dining philosophers problem specified in Maude.

```
fmod PHILOSOPHERS-TABLE is            *** functional module
  protecting NAT .                    *** import a module (natural n.)

  sorts Obj Phil List Table .         *** declare some sorts
  subsorts Obj Phil < List .          *** establish subsort relations

  op (_|_|_) : Obj Nat Obj -> Phil [ctor] .   *** constructor
  ops o ψ    : -> Obj [ctor] .
  op empty   : -> List [ctor] .
  op __      : List List -> List [ctor assoc id: empty] .
  op <_>     : List -> Table [ctor] .

  var L : List . var P : Phil .               *** declare a variable
  eq < ψ L P > = < L P ψ > .

  op initial : -> Table .
  eq initial = < (o | 0 | o) ψ (o | 1 | o) ψ (o | 2 | o) ψ > .
endfm

mod PHILOSOPHERS-DINNER is            *** system module
  protecting PHILOSOPHERS-TABLE .
  var Id : Nat .
  var X  : Obj .
  var L  : List .
  rl [left]    :        ψ (o | Id | X) => (ψ | Id | X) .
  rl [right]   :        (X | Id | o) ψ => (X | Id | ψ) .
  rl [left]    : < (o | Id | X) L ψ > => < (ψ | Id | X) L > .
  rl [release] :          (ψ | Id | ψ) => ψ (o | Id | o) ψ .
endm
```

Terms can be reduced equationally to a normal form using the `reduce` command. Rules can be applied using the `rewrite` and `frewrite` commands, which follow different fixed built-in strategies to choose which rule, which subterm, and which substitution to try first. Finally, `search` allows searching for any terms satisfying some given conditions in all possible rewriting paths from its argument [12, §5.4].

4.1 The strategy language

Effective manipulation of strategies requires expressing them in a convenient syntactical form. Since Maude is a reflective language, strategies have usually been expressed by explicitly applying rules at the metalevel. Because of its low-level, this is an awkward and error-prone method, so a strategy language was proposed [25, 15], conceived as an additional layer above functional and system specification. It has already been used in different contexts, among others [17, 24, 31, 33]. A strategy expression α can be executed to rewrite a term t using the **srewrite** t **using** α command. The language's basic component is rule application

$$\mathit{ruleLabel}\,\texttt{[}x_1\ \texttt{<-}\ t_1\texttt{,}\ \ldots\ \texttt{,}x_n\ \texttt{<-}\ t_n\texttt{]\{}\alpha_1\texttt{,}\ \ldots\ \texttt{,}\ \alpha_m\texttt{\}}$$

where rules are selected by their labels, an optional initial substitution can instantiate both rule sides before matching, and strategies between curly brackets must be provided for rules with rewriting conditions. The other basic construct is the test `match` P `s.t.` C that checks

if the subject term matches the pattern P and the condition C is satisfied. Tests come in three flavors: `match` that matches on top only, `xmatch` that can also match fragments by structural axioms, and `amatch` that matches anywhere within the term. These operators are combined by various constructs like

- concatenation α`;`β, to execute α and then β on its results;
- alternation α`|`β, which non-deterministically chooses α or β;
- iteration α`*`, which executes α zero or more consecutive times;
- the constants `idle`, to do nothing, and `fail`, to discard the current execution path;
- the conditional α `?` β `:` γ, with condition α, positive β and negative γ branches, where γ is only executed if α does not produce any result. Otherwise, β is run after any of those.

Notice that concatenation, alternation, `idle`, and `fail` are the counterparts of regular expression constructors. Derived operators are available too, like α`+` $\equiv$ α`;`α`*`, `not(`α`)` $\equiv$ α `? fail : idle`, a normalization operator α`!` $\equiv$ α`* ; not(`α`)`, etc. To control *where* rules are applied, the language counts with the `top(`α`)` operator that restricts rule applications to the top symbol, and with a subterm rewriting operator

`matchrew` $P(x_1, \ldots, x_n)$ `s.t.` C `by` x_1 `using` α_1 `,` ... `,` x_n `using` α_n

that matches the subject term against a pattern P, extracts the matched subterms, and rewrites them by means of substrategies α_1, ..., α_n in parallel. Like the `match` operator, three different versions of this operator exist. Finally, the language allows the declaration of named strategies with arguments in separate strategy modules `smod NAME is` ... `endsm`:

`strat` *slabel* [`:` *parameterTypes*] `@` *subjectType* `.`

Strategies are called *slabel*`(`t_1`,` ... `,`t_n`)` by citing its strategy name and providing the required arguments in a comma-separated list between parentheses, as a typical function invocation. They can be recursive and mutually recursive, and are defined in strategy modules with any number of (potentially conditional) definitions of the form

[`c`]`sd` *slabel*`(`*arguments*`)` `:=` *strategyExpression* [`if` C] `.`

where the strategy expression can use the variables in the left-hand side and in the condition. All strategy definitions whose left-hand side matches the call term are executed. An example is shown in Listing 2.

Listing 2 Dining philosophers strategy module.

```
smod DINNER-STRAT is
  protecting PHILOSOPHERS-DINNER .

  strats free parity @ Table .

  sd free := all ? free : idle .

  sd parity := (release
    *** The even take the left ψ first
    | (amatchrew L s.t. ψ (o | Id | o) := L /\ 2 divides Id
      by L using left)
    | left[Id <- 0]
    *** The odd take the right ψ first
    | (amatchrew L s.t. (o | Id | o) ψ := L
      /\ not (2 divides Id) by L using right)
```

```
    *** When they already have one, they take the other ψ
    | (amatchrew L s.t. (ψ | Id | o) ψ := L by L using right)
    | (matchrew M s.t. < L (o | Id | ψ) L' > := M
      by M using left[Id <- Id])
    ) ? parity : idle .
endsm
```

The DINNER-STRAT module imports the module PHILOSOPHERS-DINNER (Example 6) that it will control, and defines two recursive strategies. The first one, free, is the recursive application of any rule (all allows the application of any available rule) until it cannot be further applied. It behaves like the built-in rewrite strategy. The other strategy, parity, forces a particular order in which to take the forks, which is alternative for evens and odds, that is, for neighbors. In Section 5.1, we will see properties that are satisfied with parity but not with free.

More details on the language syntax and semantics are provided in [15, 9] and the companion web page, where the complete implementation of the strategy language is available for download.

4.2 An operational semantics for model checking

This section provides the basis to model check systems specified in Maude and controlled by its strategy language. In this situation, it is essential to know which rewriting paths are allowed by the strategy. However, the semantics of a strategy expression applied to a term has usually been given as a set of result terms [15], so that the intermediate execution states remain unknown. A small-step operational semantics is required to observe them all. One has already been given in [9] by means of a rewrite theory transformation. Still and all, it specifies a global strategic search where multiple execution paths advance in parallel, in a way that they cannot be easily isolated. Based on these, we propose a non-deterministic operational semantics whose derivations clearly denote the full rewriting paths allowed by the strategy. Some technical details have been omitted, but are available in [30].

First, we define the *strategy execution states* on which the semantics is defined. Essentially, states consist of a term t in some rewrite theory $\mathcal{R} = (\Sigma, E, R)$ and a stack s of pending strategies and variable contexts, represented by substitutions $\sigma : X \to T_{\Sigma/E}(X)$. However, the subterm rewriting operator and rewriting conditions require executing strategies in nested contexts that are represented by the "subterm" and "rewc" symbols. In summary, execution states are generated by the following grammar where x is a variable, t is a term in $\mathcal{R}$, α is a strategy expression, and ε is the empty word.

$$s ::= \varepsilon \mid \sigma s \mid \alpha s$$

$$p ::= t \mid \mathrm{subterm}(x : q, \ldots, x : q; t) \mid \mathrm{rewc}(p : q, \sigma, C, \alpha \cdots \alpha, \sigma, t, t; t)$$

$$q ::= p \,@\, s$$

Any execution state can be projected to a term by the recursive function $\mathrm{cterm}(t \,@\, s) = t$, $\mathrm{cterm}(\mathrm{subterm}(x_1 : q_1, \ldots, x_n : q_n; t) \,@\, s) = t[x_1/\mathrm{cterm}(q_1), \ldots, x_n/\mathrm{cterm}(q_n)]$ and finally $\mathrm{cterm}(\mathrm{rewc}(p : q, \sigma, C, \vec{\alpha}, \theta, r, c; t) \,@\, s) = t$. Moreover, every stack designates a variable context by looking at the top-most substitution, $\mathrm{vctx}(\varepsilon) = \mathrm{id}$, $\mathrm{vctx}(\theta s) = \theta$ and $\mathrm{vctx}(\alpha s) = \mathrm{vctx}(s)$ where id is the identity function. States of the form $t \,@\, \varepsilon$ are called *solutions* and represent successful strategy executions.

The semantics is defined by two distinct transition relations: control $\to_c$ and system $\to_s$ steps. The latter represents real transitions in the underlying system, i.e. rule rewrites, while the first does the auxiliary work to make strategies run. Among control transitions, some are devoted to handle alternation and iteration by taking non-deterministic choices,

$$t \mathbin{@} \alpha \,|\, \beta \to_c t \mathbin{@} \alpha \qquad t \mathbin{@} \alpha \,|\, \beta \to_c t \mathbin{@} \beta \qquad\qquad t \mathbin{@} \alpha\texttt{*} \to_c t \mathbin{@} \varepsilon \qquad t \mathbin{@} \alpha\texttt{*} \to_c t \mathbin{@} \alpha\alpha\texttt{*}$$

Concatenation is reduced by a rule $t \mathbin{@} \alpha \texttt{;} \beta \to_c t \mathbin{@} \alpha\beta$ that pushes α on top of β in the pending strategies stack. The rule for tests simply pops the operator on success

$$t \mathbin{@} (\texttt{match}\ P\ \texttt{s.t.}\ C)\,\theta \to_c t \mathbin{@} \theta \qquad \text{if there is } \sigma \text{ s.t. } \sigma(\theta(P)) = t \text{ and } \sigma(\theta(C)) \text{ holds}$$

where substitutions σ are extended to substitute variables within terms and conditions. The other test flavors have similar rules. There is no rule for the negative case: the execution path arrives to a deadlock state and will later be discarded. The positive case behaves like an `idle`, whose rule is $t \mathbin{@} \texttt{idle} \to_c t \mathbin{@} \varepsilon$. The semantics of conditionals is given by two rules

$$t \mathbin{@} \alpha \;\texttt{?}\; \beta \;\texttt{:}\; \gamma \to_c t \mathbin{@} \alpha\beta \qquad \frac{\text{the derivations from } t \mathbin{@} \alpha\,\theta \text{ are finite and no solution is reached}}{t \mathbin{@} (\alpha \;\texttt{?}\; \beta \;\texttt{:}\; \gamma)\,\theta \to_c t \mathbin{@} \gamma\,\theta}$$

The first rule simply tries the condition followed by the positive branch: in case α does not produce any result, β will not be executed. Then, the second strategy must be triggered to run γ. Notice that the $\to_c$ rule is undecidable in general since the negative branch condition implies deciding whether the derivations from $t \mathbin{@} \alpha\theta$ are all terminating.

Strategy calls are handled with rule instances of the form

$$t \mathbin{@} sl(p_1, \ldots, p_n)\,\theta \to_c t \mathbin{@} \delta\sigma\theta \qquad \text{for any matching } \sigma \text{ of } (t_i)_{i=1}^n \text{ in } (p_i)_{i=1}^n \text{ s.t. } \sigma(C) \text{ holds}$$

for each strategy definition `csd` sl`(`t_1`, ..., `t_n`) :=` δ `if` C (or its unconditional version). When a strategy call finishes, its variable context is popped $t \mathbin{@} \theta \to_c t \mathbin{@} \varepsilon$.

The mission of the execution stack is holding pending strategies and also active call contexts. Only the top element determines the possible next steps, but the current variable context may be determined by a substitution buried by multiple strategies inside the stack. Rules have been defined for states with almost empty stacks, but they can be easily extended from the bottom as long as the variable context is preserved.

$$t \mathbin{@} s = t \mathbin{@} s\,\mathrm{id} \qquad\qquad \frac{t \mathbin{@} s\,\theta \to_\bullet t' \mathbin{@} s'\,\theta}{t \mathbin{@} s\,s_0 \to_\bullet t \mathbin{@} s' s_0} \text{ if } \mathrm{vctx}(s_0) = \theta$$

where $\to_\bullet$ can be replaced by both $\to_s$ and $\to_c$.

The `matchrew` rewrites matched subterms independently via the subterm execution states,

$$\frac{q_i \to_s^c q_i'}{\mathrm{subterm}(\ldots, x_i : q_i, \ldots; t) \mathbin{@} s \to_s^c \mathrm{subterm}(\ldots, x_i : q_i', \ldots; t) \mathbin{@} s}$$

These structured states are created with the rule

$$\begin{aligned} t \mathbin{@} (\texttt{matchrew}\ P(x_1, \ldots, x_n)\ \texttt{s.t.}\ C\ \texttt{by}\ x_1\ \texttt{using}\ \alpha_1, \ldots, x_n\ \texttt{using}\ \alpha_n)\,\theta \\ \to_c \mathrm{subterm}(x_1 : \sigma(x_1) \mathbin{@} \alpha_1\,\rho, \ldots, x_n : \sigma(x_n) \mathbin{@} \alpha_n\,\rho, \ldots) \mathbin{@} \theta \end{aligned}$$

for any matching σ of $\theta(P)$ in t such that $\sigma(\theta(C))$ holds, and where $\rho(x) = \sigma(x)$ if $\sigma(x) \neq x$ and $\theta(x)$ otherwise. And they are resolved, once its subterms have arrived to solutions, with

$$\mathrm{subterm}(x_1 : t_1 \mathbin{@} \varepsilon, \ldots, x_n : t_n \mathbin{@} \varepsilon; t) \mathbin{@} s \to_c t[x_1/t_1, \ldots, x_n/t_n] \mathbin{@} s$$

Finally, system transitions $\to_s$ are generated by rule applications. The execution of a maybe conditional rule without rewriting fragments is

$$t \,@\, rl[x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n]\,\theta \to_s t[p/\sigma(\rho(r))] \,@\, \theta$$

if for a position p within t and for a rule $l \to r$, there is a matching σ such that $t_p = \sigma(\rho(l))$ and $\sigma(\rho(C))$ holds, where ρ is the initial substitution that maps x_i to $\theta(t_i)$. If the rule application is surrounded by the `top` modifier, it is only applied on top. For rules with rewriting conditions l `=>` r, substrategies must be provided between curly brackets and these must be used to rewrite its condition fragments. This is achieved using a structured execution state similar to the subterm construct, where the nested state q executes the corresponding strategy in the rewriting fragment initial term.

$$t \,@\, rl[\ldots, x_i \leftarrow t_i, \ldots]\{\alpha_1, \ldots, \alpha_m\}\,\theta \to_c \mathrm{rewc}(t_r : \sigma(t_l) \,@\, \alpha_1\theta, \sigma, C, \alpha_2 \cdots \alpha_m, \theta, r, c; t)$$

for any rule rl with condition $C_0 \wedge t_l \texttt{=>} t_r \wedge C$ where C_0 is an equational condition, and any matching substitution σ and matching context c such that $\sigma(C_0)$ holds. Like in the previous case, the rule is first instantiated with the given initial substitution. The right-hand side of the rule r is kept to do the actual rewriting once the conditions have been checked:

$$\mathrm{rewc}(p : t' \,@\, \varepsilon, \sigma, C_0, \varepsilon, \theta, r, c; t) \to_s c(\sigma'(r))$$

where the substitution σ' extends σ by matching t' against $\sigma(p)$ and satisfies the equational condition $\sigma'(C_0)$. If the remaining condition contains more rewriting fragments, these are tried one after another accumulating variable bindings:

$$\mathrm{rewc}(p : t' \,@\, \varepsilon, \sigma, C_0 \wedge t_l \texttt{=>} t_r \wedge C, \alpha\vec{\alpha}, \theta, r, c; t) \to_c \mathrm{rewc}(t_r : \sigma'(t_l) \,@\, \alpha\,\theta, \sigma', C, \vec{\alpha}, \theta, r, c; t)$$

The rewc state follows the transitions of the inner state,

$$\frac{q \to_\bullet q'}{\mathrm{rewc}(p : q, \sigma, C, \vec{\alpha}, \theta, r, c; t) \to_c \mathrm{rewc}(p : q', \sigma, C, \vec{\alpha}, \theta, r, c; t)}$$

However, transitions inside this nested state are always control transitions in the outer one, because they are not applied to the subject term.

Finally, this semantics can be used to define the translation of the strategy expression α to an abstract strategy $E(\alpha)$ in $(T_{\Sigma/E}, \to^1_{\mathcal{R}})$ where $T_{\Sigma/E}$ is the initial term algebra and $\to^1_{\mathcal{R}}$ the one-step rewrite relation of the rewrite theory $\mathcal{R}$. The derived relation $\twoheadrightarrow = \to_c^* \circ \to_s$, a single system transition preceded with all the necessary strategic preparation, has the property that $q \twoheadrightarrow q'$ implies $\mathrm{cterm}(q) \to^1_{\mathcal{R}} \mathrm{cterm}(q')$.

▶ **Definition 7.** *For a strategy expression α and a set of initial states I, we define*

$$\begin{aligned} E(\alpha) :=& \{t\,\mathrm{cterm}(q_1)\cdots\mathrm{cterm}(q_n)\cdots : t \,@\alpha \twoheadrightarrow q_1 \twoheadrightarrow \cdots \twoheadrightarrow q_n \twoheadrightarrow \cdots, t \in I\} \\ &\cup\ \{t\,\mathrm{cterm}(q_1)\cdots\mathrm{cterm}(q_n) : t \,@\alpha \twoheadrightarrow q_1 \twoheadrightarrow \cdots \twoheadrightarrow q_n \to_c^* t_n \,@\varepsilon, t \in I\} \end{aligned}$$

Observe that finite derivations end with solutions, perhaps modulo some control transitions. The strategy $E(\alpha)$ is intensional and can be encoded in the ARS $\mathcal{A}' = (\mathcal{XS}, \twoheadrightarrow, \{t \,@\, \alpha : t \in I\})$ with final states $F = \{q \in \mathcal{XS} : \exists\, t \in T_{\Sigma/E} \quad q \to_c^* t \,@\, \varepsilon\}$, and $d = \mathrm{cterm}$ the descent function. $\mathcal{XS}$, the set of execution states, is always an infinite set, but we can restrict to the reachable execution states from the initial ones. For model checking to be decidable, the ARS needs to be finite. Some sufficient conditions can be established:

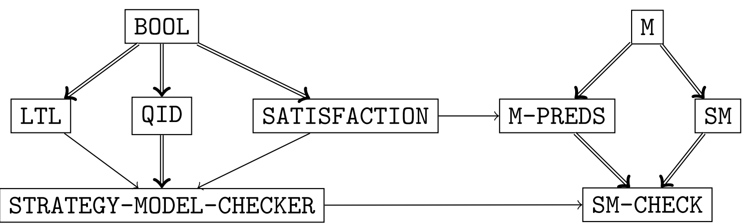


Figure 3 Structure of the strategy model checker modules.

▶ **Proposition 8.** *Given a term $t \in T_{\Sigma/E}$ and a strategy expression α, if the reachable terms from $t\,@\,\alpha$ are finitely many, and the recursive strategy calls are tail recursive and their call arguments only take a finite number of values, $\twoheadrightarrow$ is decidable and the set of reachable execution states is finite.*

Both premises are reasonable, since they bound the number of state and strategy combinations that may appear during execution. This also implies the decidability of the $\twoheadrightarrow$ relation, whose threats are the negative branch of the conditional combinator and rewriting conditions. We understand by *reachable terms* all the elements of $T_{\Sigma/E}$ that occur while executing the strategy, while rewriting both the state and the rewriting conditions of rules. It is easy to observe that expressions without iterations and recursive calls never produce an infinite number of execution states, but they are not usually interesting.

Often, this sufficient condition holds and is checked easily, like in the example strategies of Listing 2. In the `free` and `parity` definitions, all strategy calls are tail recursive and do not take parameters. The reachable terms from `initial` are finitely many since, even by unrestricted rewriting, only 3^3 *tables* are reachable, as shown by counting the possible positions of the forks.

5 The Maude strategy-aware model checker

Following the principles of Section 3 and the strategy language semantics, the new Maude strategy-aware model checker was programmed in C++ as an extension of the already existing explicit-state LTL model checker [16]. Both have a similar interface [12, §10] and are accessed using separate special operators declared in Maude itself.

The built-in modules of the model checker appear on the left of Figure 3. All but `STRATEGY-MODEL-CHECKER` are shared with the standard model checker. The right side of the figure shows the typical structure of the user specification of the model and properties. The user must:

1. Specify the model in a system module `M`, and define strategies to control `M` in a strategy module `SM`.
2. In a protecting[2] extension of `M`, say `M-PREDS`, choose the sort of the model states, making it a subsort of the `State` sort declared in `SATISFACTION`, declare atomic propositions as operators of type `Prop`, and define the satisfaction relation `|=` for all of them.

```
fmod SATISFACTION is
  protecting BOOL . sorts State Prop .
  op _|=_ : State Prop -> Bool .
endfm
```

[2] Protecting means that it does not alter the signature, equations and rules of the types defined in `M`.

```
mod M-PREDS is
  protecting M .
  including SATISFACTION .
  subsort Foo < State .
  op p : -> Prop .
  eq F:Foo |= p = ... .
endm
```

3. Declare a strategy module, say `SM-CHECK`, to combine the model `M` with the property specification in `M-PREDS` and the strategies `SM`. Import `STRATEGY-MODEL-CHECKER` too.

```
smod SM-CHECK is
  protecting M-PREDS .
  protecting SM .
  including STRATEGY-MODEL-CHECKER .
endsm
```

Once this is done, model checking is invoked using the operator

```
op modelCheck : State Formula Qid QidList
  ~> ModelCheckerResult [special (...)] .
```

which receives an initial state, an LTL formula as defined in the `LTL` module [12, §10], and a strategy identifier, which must correspond to a strategy without parameters defined in the module. The last argument is an optional list of *opaque* strategy names: when a strategy in this list is called, instead of the transitions occurring during the strategy execution, the model checker will see direct transitions to its results. This optional feature produces traces that do not fit in the base `M` model, but allows model checking coarse-grain and fine-grain models with little changes.

The result is either `true` if the model satisfies the specification, or a counterexample, expressed as a cycle of rewriting steps and a path to it, if it does not. Additionally, the model checker optionally outputs an extended dump, from which graphical representations of the system automaton and counterexamples can be generated using an auxiliary program. The model checker, the auxiliary program, additional documentation, and various examples can be downloaded at `http://maude.sip.ucm.es/strategies`.

The fundamentals of the strategy-aware model checker are given by the small-step operational semantics of Section 4.2 and the fundamentals of the original model checker. The model controlled by a strategy α from an initial term $t \in T_{\Sigma/E}$ can be encoded in the Kripke structure

$$\mathcal{K} = (\mathcal{XS}, \twoheadrightarrow, \{t \mathbin{@} \alpha\}, AP_\Pi, L_\Pi \circ \text{cterm}),$$

where atomic propositions are defined as in the standard model checker, for Π the signature of `M-PREDS` and D its set of equations,

$$AP_\Pi := \{\ \theta(p(x_1, \dots, x_n)) \mid p \in \Pi, \theta \text{ ground substitution }\}.$$

The labeling function $L_\Pi : T_{\Sigma/E} \to \mathcal{P}(AP_\Pi)$ is given by

$$L_\Pi([t]) := \{\theta(p(x_1, \dots, x_n)) \in AP_\Pi \mid (E \cup D) \vdash\ t \vDash \theta(p(x_1, \dots, x_n)) = \mathit{true}\}.$$

For the model checking to be decidable the conditions in [12, §10.3] for the standard one must be fulfilled, and additionally the reachable execution states must be finitely many.

5.1 Example: dining philosophers

In this section, we resume the dining philosophers problem (Example 6) to model check some properties with different strategies. Although the example is presented with three philosophers, it can be instantiated with many more. We already know that some unwanted situations may appear during the dinner: a philosopher may starve or, even worse, none of them could be able to eat. First, we express the collection of properties "the philosopher n eats" as atomic propositions and prepare the model checker input data.

Listing 3 Atomic proposition for the dining philosophers example.

```
mod DINNER-PREDS is
  protecting PHILOSOPHERS-DINNER .    *** From Example 6 (Listing 1)
  including SATISFACTION .

  subsort Table < State .
  op eats : Nat -> Prop [ctor] .

  var  Id  : Nat .
  vars L M : List .

  eq < L (ψ | Id | ψ) M > |= eats(Id) = true .
  eq < L > |= eats(Id) = false [owise] .        *** otherwise
endm
```

Then, we put together and include the built-in model checker module.

```
smod DINNER-CHECK is
  protecting DINNER-PREDS .
  protecting DINNER-STRAT .       *** From Listing 2
  including STRATEGY-MODEL-CHECKER .
  including MODEL-CHECKER .       *** the standard model checker too
endsm
```

Now, we can check that the property $\Box \Diamond (eats(0) \lor eats(1) \lor eats(2))$ (no *deadlock*) does not hold for free rewriting, either using the standard model checking or the strategy-aware one with the `free` strategy, but it does hold when using the `parity` strategy.

```
Maude> red modelCheck(initial,
             [] <> (eats(0) \/ eats(1) \/ eats(2))) .
ModelCheckerSymbol: Examined 4 system states.
rewrites: 43 in 4ms cpu (0ms real) (10750 rewrites/second)
result ModelCheckResult: counterexample(
  {< (o | 0 | o) ψ (o | 1 | o) ψ (o | 2 | o) ψ >,'left}
  {< (ψ | 0 | o) ψ (o | 1 | o) ψ (o | 2 | o) >, 'left}
  {< (ψ | 0 | o) (ψ | 1 | o) ψ (o | 2 | o) >,'left},
  {< (ψ | 0 | o) (ψ | 1 | o) (ψ | 2 | o) >,deadlock})

Maude> red modelCheck(initial,
             [] <> (eats(0) \/ eats(1) \/ eats(2)), 'parity) .
StrategyModelCheckerSymbol: Examined 12 system states.
rewrites: 159 in 0ms cpu (2ms real) (~ rewrites/second)
result Bool: true
```

However, `parity` does not guarantee that all of them eat, because the property $\Diamond\, eats(0)$ does not hold. The model checker presents a counterexample, where the philosopher number two takes both forks and releases them in a loop, while the rest keep inactive:

```
Maude> red modelCheck(initial, <> eats(0), 'parity) .
rewrites: 55 in 0ms cpu (0ms real) (~ rewrites/second)
result ModelCheckResult: counterexample(nil,
  {< (o | 0 | o) ψ (o | 1 | o) ψ (o | 2 | o) ψ >,'left}
  {< (o | 0 | o) ψ (o | 1 | o) (ψ | 2 | o) ψ >,'right}
  {< (o | 0 | o) ψ (o | 1 | o) (ψ | 2 | ψ) >,'release})
```

In order to ensure that all of them eat, the strategy should act as a referee. A succinct and direct solution is fixing turns, like in the following strategy:

```
sd turns(K, N) := left[Id <- K] ; right[Id <- K] ; release ;
            turns(s(K) rem N, N) .
sd turns := turns(0, 3) .
```

If the number of diners is greater, a more parallel version can be written allowing n div 2 philosophers to eat at the same time. By model checking with `turns`, we obtain

```
Maude> red modelCheck(initial,
  [] (<> eats(0) /\ <> eats(1) /\ <> eats(2)), 'turns) .
rewrites: 131 in 0ms cpu (1ms real) (~ rewrites/second)
result Bool: true
```

5.2 Implementation notes

We have programmed the new model checker in C++ as part of a modified version of the Maude interpreter that includes full strategy language support. The implementation reuses both the existing explicit-state LTL model checker and the existing infrastructure for strategic execution [15], which we have completed to support strategy modules and the `matchrew` operator. This infrastructure is based on a collection of *tasks*, which reflect continuations and call frames, and *processes* that are in charge of finding matches, applying rules, etc. The already existing model checker follows the automata-theoretic approach [11, §4] based on testing the emptiness of the language recognized by the synchronous product of the model and the negation of the linear property as Büchi automata. The model automaton is built specifically for the system controlled by the strategy, while the LTL to Büchi automaton translation and the nested depth-first algorithm are reused.

Each model state corresponds to a strategy execution state in the proposed semantics, and stores some identifying information and a list of processes from which successors are obtained on-the-fly when requested. Control operations $\to_c$ are handled as in usual execution, but rule rewrites trigger the commitment of a new state. Different techniques are used to detect cycles and already visited states in order to reuse previous work.

6 Conclusions and future work

Strategies and languages to express them are useful resources to specify restrictions and global control in rewriting systems, following the *separation of concerns* principle. This approach has already been used to specify deduction procedures, semantics of programming languages, chemical and biological processes, ... Such models need to be formally verified and analyzed. In this paper, we show that model checking has a natural definition in this context, and we tell how to effectively model check specifications for the Maude strategy language, by means of a small-step operational semantics that emphasizes rewriting sequences.

This procedure can be applied to other strategy representations. A model checker for systems specified in Maude and controlled by its strategy language has been implemented in C++ as an extension of the existing explicit-state LTL model checker, which can be downloaded from `http://maude.sip.ucm.es/strategies`. It has already been tested with classical examples and its performance is comparable to the original model checker.

The ongoing work comprises the study of branching-time properties and other theoretical aspects, as well as the development of examples to exploit and test the performance of the tool. The model checker can also be improved by providing clearer counterexamples with more information on the strategy execution, and updating the inherited LTL-to-automaton algorithm to more recent and efficient proposals [3, 22].

References

1. Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002. doi:10.1145/585265.585270.
2. Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998. doi:10.1017/CBO9781139172752.
3. Tomás Babiak, Mojmír Kretínský, Vojtech Rehák, and Jan Strejcek. LTL to Büchi automata translation: Fast and more deterministic. In Cormac Flanagan and Barbara König, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 18th International Conference, TACAS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, volume 7214 of *LNCS*, pages 95–109. Springer, 2012. doi:10.1007/978-3-642-28756-5_8.
4. Emilie Balland, Paul Brauner, Radu Kopetz, Pierre-Etienne Moreau, and Antoine Reilles. Tom: Piggybacking Rewriting on Java. In Franz Baader, editor, *Term Rewriting and Applications, 18th International Conference, RTA 2007, Paris, France, June 26-28, 2007, Proceedings*, volume 4533 of *LNCS*, pages 36–47. Springer, 2007. doi:10.1007/978-3-540-73449-9_5.
5. H.P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 131. North Holland, 2nd edition, 2014.
6. Massimo Bartoletti, Maurizio Murgia, Alceste Scalas, and Roberto Zunino. Modelling and Verifying Contract-Oriented Systems in Maude. In Santiago Escobar, editor, *Rewriting Logic and Its Applications - 10th International Workshop, WRLA 2014, Held as a Satellite Event of ETAPS, Grenoble, France, April 5-6, 2014, Revised Selected Papers*, volume 8663 of *LNCS*, pages 130–146. Springer, 2014. doi:10.1007/978-3-319-12904-4_7.
7. Peter Borovanský, Claude Kirchner, Hélène Kirchner, Pierre-Etienne Moreau, and Christophe Ringeissen. An Overview of ELAN. In Claude Kirchner and Hélène Kirchner, editors, *Proceedings of the Second International Workshop on Rewriting Logic and its Applications, WRLA'98, Pont-à-Mousson, France, September 1-4, 1998*, volume 15 of *ENTCS*, pages 55–70. Elsevier, 1998. doi:10.1016/S1571-0661(05)82552-6.
8. Tony Bourdier, Horatiu Cirstea, Daniel J. Dougherty, and Hélène Kirchner. Extensional and Intensional Strategies. In Maribel Fernández, editor, *Proceedings Ninth International Workshop on Reduction Strategies in Rewriting and Programming, WRS 2009, Brasilia, Brazil, 28th June 2009*, volume 15 of *EPTCS*, pages 1–19, 2009. doi:10.4204/EPTCS.15.1.
9. Christiano Braga and Alberto Verdejo. Modular Structural Operational Semantics with Strategies. In Rob van Glabbeek and Peter D. Mosses, editors, *Proceedings of the Third Workshop on Structural Operational Semantics, SOS 2006, Bonn, Germany, August 26, 2006*, volume 175(1) of *ENTCS*, pages 3–17. Elsevier, 2007. doi:10.1016/j.entcs.2006.10.024.
10. Martin Bravenboer, Karl Trygve Kalleberg, Rob Vermaas, and Eelco Visser. Stratego/XT 0.17. A language and toolset for program transformation. *Science of Computer Programming*, 72(1-2):52–70, 2008. doi:10.1016/j.scico.2007.11.003.
11. Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors. *Handbook of Model Checking*. Springer, 2018. doi:10.1007/978-3-319-10575-8.

12 Manuel Clavel, Francisco Durán, Steven Eker, Santiago Escobar, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn Talcott. *Maude Manual v2.7.1*, July 2016. URL: `http://maude.lcc.uma.es/manual271/maude-manual.html`.

13 Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn L. Talcott. *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *LNCS*. Springer, 2007. `doi:10.1007/978-3-540-71999-1`.

14 Stefan Edelkamp, Stefan Leue, and Alberto Lluch-Lafuente. Directed explicit-state model checking in the validation of communication protocols. *International Journal on Software Tools for Technology Transfer*, 5(2-3):247–267, 2004. `doi:10.1007/s10009-002-0104-3`.

15 Steven Eker, Narciso Martí-Oliet, José Meseguer, and Alberto Verdejo. Deduction, Strategies, and Rewriting. In Myla Archer, Thierry Boy de la Tour, and César Muñoz, editors, *Proceedings of the 6th International Workshop on Strategies in Automated Deduction, STRATEGIES 2006, Seattle, WA, USA, August 16, 2006*, volume 174(11) of *ENTCS*, pages 3–25. Elsevier, 2007. `doi:10.1016/j.entcs.2006.03.017`.

16 Steven Eker, José Meseguer, and Ambarish Sridharanarayanan. The Maude LTL Model Checker. In Fabio Gadducci and Ugo Montanari, editors, *Proceedings of the Fourth International Workshop on Rewriting Logic and its Applications, WRLA 2002, Pisa, Italy, September 19-21, 2002*, volume 71 of *ENTCS*, pages 162–187. Elsevier, 2004. `doi:10.1016/S1571-0661(05)82534-4`.

17 Mercedes Hidalgo-Herrero, Alberto Verdejo, and Yolanda Ortega-Mallén. Using Maude and Its Strategies for Defining a Framework for Analyzing Eden Semantics. In Sergio Antoy, editor, *Proceedings of the Sixth International Workshop on Reduction Strategies in Rewriting and Programming, WRS 2006, Seattle, WA, USA, August 11, 2006*, volume 174(10) of *ENTCS*, pages 119–137. Elsevier, 2007. `doi:10.1016/j.entcs.2007.02.051`.

18 C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

19 Hélène Kirchner. Rewriting Strategies and Strategic Rewrite Programs. In Narciso Martí-Oliet, Peter Csaba Ölveczky, and Carolyn L. Talcott, editors, *Logic, Rewriting, and Concurrency - Essays dedicated to José Meseguer on the Occasion of His 65th Birthday*, volume 9200 of *LNCS*, pages 380–403. Springer, 2015. `doi:10.1007/978-3-319-23165-5_18`.

20 Saul Kripke. A Completeness Theorem in Modal Logic. *Journal of Symbolic Logic*, 24(1):1–14, 1959. `doi:10.2307/2964568`.

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

22 Weiwei Li, Shuanglong Kan, and Zhiqiu Huang. A Better Translation From LTL to Transition-Based Generalized Büchi Automata. *IEEE Access*, 5:27081–27090, 2017. `doi:10.1109/ACCESS.2017.2773123`.

23 Si Liu, Muntasir Raihan Rahman, Stephen Skeirik, Indranil Gupta, and José Meseguer. Formal Modeling and Analysis of Cassandra in Maude. In Stephan Merz and Jun Pang, editors, *Formal Methods and Software Engineering - 16th International Conference on Formal Engineering Methods, ICFEM 2014, Luxembourg, Luxembourg, November 3-5, 2014. Proceedings*, volume 8829 of *LNCS*, pages 332–347. Springer, 2014. `doi:10.1007/978-3-319-11737-9_22`.

24 Narciso Martí-Oliet, Miguel Palomino, and Alberto Verdejo. Strategies and simulations in a semantic framework. *Journal of Algorithms*, 62(3-4):95–116, 2007. `doi:10.1016/j.jalgor.2007.04.002`.

25 Narciso Martí-Oliet, José Meseguer, and Alberto Verdejo. Towards a Strategy Language for Maude. In Narciso Martí-Oliet, editor, *Proceedings of the Fifth International Workshop on Rewriting Logic and its Applications, WRLA 2004, Barcelona, Spain, March 27-April 4, 2004*, volume 117 of *ENTCS*, pages 417–441. Elsevier, 2004. `doi:10.1016/j.entcs.2004.06.020`.

26 José Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992. `doi:10.1016/0304-3975(92)90182-F`.

27 Fabio Mogavero, Aniello Murano, Giuseppe Perelli, and Moshe Y. Vardi. Reasoning About Strategies: On the Model-Checking Problem. *ACM Transactions in Computational Logic*, 15(4):34:1–34:47, 2014. doi:10.1145/2631917.

28 Martin R. Neuhäußer and Thomas Noll. Abstraction and Model Checking of Core Erlang Programs in Maude. In Grit Denker and Carolyn Talcott, editors, *Proceedings of the 6th International Workshop on Rewriting Logic and its Applications, WRLA 2006, Vienna, Austria, April 1-2, 2006*, volume 176 (4) of *ENTCS*, pages 147–163. Elsevier, 2007. doi:10.1016/j.entcs.2007.06.013.

29 Amir Pnueli. The Temporal Logic of Programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.32.

30 Rubén Rubio, Narciso Martí-Oliet, Isabel Pita, and Alberto Verdejo. Model checking strategy-controlled rewriting systems (extended version). Technical Report 02/19, Departamento de Sistemas Informáticos y Computación, Universidad Complutense de Madrid, 2019. URL: http://maude.sip.ucm.es/strategies/tr0219.pdf.

31 Gustavo Santos-García and Miguel Palomino. Solving Sudoku Puzzles with Rewriting Rules. In Grit Denker and Carolyn Talcott, editors, *Proceedings of the 6th International Workshop on Rewriting Logic and its Applications, WRLA 2006, Vienna, Austria, April 1-2, 2006*, volume 176 (4) of *ENTCS*, pages 79–93. Elsevier, 2007. doi:10.1016/j.entcs.2007.06.009.

32 Terese. *Term Rewriting Systems*. Cambridge University Press, 2003.

33 Alberto Verdejo and Narciso Martí-Oliet. Basic completion strategies as another application of the Maude strategy language. In Santiago Escobar, editor, *Proceedings 10th International Workshop on Reduction Strategies in Rewriting and Programming, WRS 2011, Novi Sad, Serbia, 29 May 2011*, volume 82 of *EPTCS*, pages 17–36, 2011. doi:10.4204/EPTCS.82.2.