9th International Conference on Formal Structures for Computation and Deduction

FSCD 2024, July 10-13, 2024, Tallinn, Estonia

Edited by

Jakob Rehof

Editors

Jakob Rehof
TU Dortmund University, Germany
Lamarr Institute for Machine Learning and Artificial Intelligence, Dortmund, Germany
jakob.rehof@cs.tu-dortmund.de

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

ISBN 978-3-95977-323-2

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/978-3-95977-323-2.

Publication date
July, 2024

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

Digital Object Identifier: 10.4230/LIPIcs.FSCD.2024.0

LIPIcs – Leibniz International Proceedings in Informatics

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

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

Invited Talks

Regular Papers

9th International Conference on Formal Structures for Computation and Deduction (FSCD 2024).
Editor: Jakob Rehof
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

This volume contains the proceedings of the 9th International Conference on Formal Structures for Computation and Deduction (FSCD 2024), which was held July 10-13, 2024 in Tallinn, Estonia. FSCD 2024 was co-located with the 51st EATCS International Colloquium on Automata, Languages, and Programming (ICALP), which was held July 8-12, 2024 in Tallinn, Estonia, and with the 39th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), which was held July 8-11, 2024 in Tallinn, Estonia.

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

The FSCD 2024 program featured five invited talks, two of which were joint between ICALP, LICS, and FSCD. The joint invited speakers were Edith Elkind (University of Oxford, UK) and Stephanie Weirich (University of Pennsylvania, USA). The FSCD invited speakers were Delia Kesner (Université Paris Cité, France), Bettina Könighofer (Graz University of Technology, Austria), and Sebastian Ullrich (Lean Focused Research Organisation, USA). The contributions of the FSCD invited speakers are included in these proceedings.

The Program Committee of FSCD 2024 consisted of 28 members from 14 countries. FSCD 2024 received 57 submissions with contributing authors from 19 countries. Every submitted paper was reviewed by at least three PC members with the help of in total 75 external reviewers. The reviewing process, which included a rebuttal phase, took place over a period of nine weeks. A total of 30 papers were accepted for publication and are included in these proceedings. The EasyChair conference management system has been a very useful tool in all phases of the work of the Program Committee.

The Program Committee awarded the FSCD 2024 Best Paper Award by Junior Researchers to Victor Sannier from Univ. Lille, CNRS, Inria, France for his paper 'A Linear Type System for L^p-Metric Sensitivity Analysis'.

In addition to the main conference, nine workshops were held before the conference:

- AATG 2024: Algorithmic Aspects of Temporal Graphs VII
- GETCO 2024: 13th International Workshop on Geometric and Topological Methods in Computer Science
- ITRS 2024: 11th Workshop on Intersection Types and Related Systems
- IWC 2024: 13th International Workshop on Confluence
- LearnAut 2024: Learning and Automata
- LFMTP 2024: Logical Frameworks and Meta Languages: Theory and Practice
- LMW 2024: 11th Logic Mentoring Workshop
- MSFP 2024: Mathematically Structured Functional Programming
- PAAW 2024: Parameterized Approximation Algorithms Workshop
- PACS 2024: Parameterized Algorithms and Constraint Satisfaction
- SmP 2024: Structure meets Power
- TAT 2024: Trends in Arithmetic Theories
- TLLA 2024: Eighth International Workshop on Trends in Linear Logic and Applications
- Women in Logic 2024

9th International Conference on Formal Structures for Computation and Deduction (FSCD 2024).
Editor: Jakob Rehof
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

This volume of FSCD 2024 is published in the LIPIcs series under a Creative Commons license: online access is free to all papers and authors retain rights over their contributions. We thank the Leibniz Center for Informatics at Schloss Dagstuhl, and in particular Michael Wagner and Michael Didas, for their prompt and helpful replies to our questions regarding the production of these proceedings, for their flexibility, and for their user-friendly submission system.

A succesful conference is the result of joint work of many people. On behalf of the Program Committee, I sincerely thank all authors of submitted papers for considering FSCD as a venue for their work. I thank the Program Committee and the external reviewers for their indispensable contribution, providing careful and constructive review and evaluation of the submitted papers. I thank all invited speakers for enriching ICALP, LICS, and FSCD with their talks. On behalf of the FSCD Program Committee, I thank all workshop organizers for making an essential contribution to the program and atmosphere of the meeting, and I thank the Steering Committee Workshop Chair Cynthia Kop and the Conference Workshop Chair Luigi Liquori for their efforts. Warm thanks to the Conference Chair of FSCD 2024, Niccolò Veltri and his colleagues at Tallinn University of Technology for the excellent organization of the conference. I also wish to thank Pawel Sobocinski (Tallinn University of Technology) for his support in the joint organization of ICALP, LICS, and FSCD in Tallinn.

I am very grateful to all the members of the Steering Committee of FSCD for their valuable and helpful guidance in setting up the meeting, and for ensuring that FSCD will remain a succesful conference. Let me thank in particular the Steering Committee chair Herman Geuvers and the Steering Committee Publicity Chair Carsten Fuhs for their support during the preparation of the conference. I am grateful to previous Program Chairs of FSCD for sharing their experience with me at various stages, including Sandra Alves, Amy Felty, Delia Kesner, Naoki Kobayashi, and Femke van Raamsdonk. Finally, I thank all participants of the conference for creating a lively and interesting event.

Jakob Rehof
Program Chair of FSCD 2024

Committees

Program Committee

Thorsten Altenkirch	University of Nottingham, UK
Sandra Alves	University of Porto, Portugal
Takahito Aoto	Niigata University, Japan
Mauricio Ayala-Rincon	Universidade de Brasília, Brazil
Thierry Coquand	Chalmers University of Technology, Sweden
Alejandro Diaz-Caro	UNQ, UBA, and CONICET, Argentina
Claudia Faggian	CNRS, Université de Paris, France
Simon Gay	University of Glasgow, UK
Silvia Ghilezan	University of Novi Sad, Serbia
Jakob Grue Simonsen	University of Copenhagen, Denmark
Cezary Kaliszyk	University of Innsbruck, Austria
Ambrus Kaposi	Eötvös Loránd University, Budapest, Hungary
Dexter Kozen	Cornell University, USA
Dominique Larchey-Wendling	CNRS, Loria, France
Marina Lenisa	University of Udine, Italy
Sonia Marin	University of Birmingham, UK
Naoki Nishida	Nagoya University, Japan
Christine Paulin-Mohring	Université Paris-Sud, France
Pierre-Marie Pédrot	Inria Rennes-Bretagne-Atlantique, France
Elaine Pimentel	University College London, UK
Jakob Rehof (Chair)	TU Dortmund University, Germany
Simona Ronchi della Rocca	University of Torino, Italy
Sylvain Schmitz	Université Paris Cité, France
Aleksy Schubert	University of Warsaw, Poland
Kathrin Stark	Heriot-Watt University, Edinburgh, UK
Lutz Straßburger	Inria Saclay, France
Tachio Terauchi	Waseda University, Japan
Sarah Winkler	Free University of Bozen-Bolzano, Italy

9th International Conference on Formal Structures for Computation and Deduction (FSCD 2024).
Editor: Jakob Rehof

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

Conference Chair

Niccolò Veltri	Tallinn University of Technology, Estonia

Workshop Chair

Luigi Liquori	Inria, France

Steering Committee Workshop Chair

Cynthia Kop	Radboud University Nijmegen, The Netherlands

Publicity Chair

Carsten Fuhs	Birkbeck, University of London, UK

Steering Committee

Patrick Baillot	CNRS, Université de Lille, France
Alejandro Díaz-Caro	UNQ, UBA, and CONICET, Argentina
Amy Felty	University of Ottawa, Canada
Carsten Fuhs	Birkbeck, University of London, UK
Marco Gaboardi	Boston University, USA
Herman Geuvers (Chair)	Radboud University Nijmegen, The Netherlands
Jürgen Giesl	RWTH Aachen University, Germany
Delia Kesner	Université Paris Cité, France
Naoki Kobayashi	University of Tokyo, Japan
Cynthia Kop	Radboud University Nijmegen, The Netherlands
Luigi Liquori	Inria, France
Giulio Manzonetto	Université Paris-Nord, France
Daniele Nantes	Imperial College London, UK & University of Brasília, Brazil
Femke van Raamsdonk	Vrije Universiteit Amsterdam, The Netherlands

External Reviewers

Blair Archibald
Ali Assaf
Martin Avanzini
Davide Barbarossa
João Barbosa
Pablo Barenbaum
Stefano Berardi
Małgorzata Biernacka
Ulrik Buchholtz
David Cerna
Stefan Ciobaca
Robin Cockett
Bruno Courcelle
Anupam Das
Abhishek De
Pietro Di Gianantonio
Andrej Dudenhefner
Rachid Echahed
Burak Ekici
Pedro Jorge Fernandes Ângelo
Francesco Gavazzo
Lorenzo Gheri
Amar Hadzihasanovic
Nao Hirokawa
Federico Holik
Furio Honsell
Kuen-Bang Hou Favonia
Justin Hsu
Jan Jakubuv
Jeroen Ketema
Maja Kirkeby
Ambroise Lafont
Paul Blain Levy
Philippe Malbos
Octavio Malherbe
Matteo Manighetti
Giulio Manzonetto
Marius Mikučionis
Masaki Nakamura
Akio Nakata
Koji Nakazawa
Jacob Neumann
Lê Thành Dũng Nguyễn
Johannes Niederhauser
Andreas Nuyts
Federico Olimpieri
Luke Ong
Luca Paolini
Dusko Pavlovic
Carla Piazza
Mario Piazza
Paolo Pistone
Romain Péchoux
Jorge A. Pérez
Florian Rabe
Camilo Rocha
Luca Roversi
Masahiko Sakai
Haruhiko Sato
Ivan Scagnetto
Ulrich Schöpp
Jonathan Sterling
Claude Stolze
Pavle Subotic
Matteo Tesi
Benoît Valiron
Iris van der Giessen
Gabriele Vanoni
Renaud Vilmart
Andrew Wagner
Quanlong Wang
Szumi Xie
Akihisa Yamada
Tetsuo Yokoyama
Hans Zantema

9th International Conference on Formal Structures for Computation and Deduction (FSCD 2024).
Editor: Jakob Rehof
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

List of Authors

Samson Abramsky (28)
Department of Computer Science, University College London, London, UK

Beniamino Accattoli (23, 24)
Inria & LIX, Ecole Polytechnique, UMR 7161, France

Benedikt Ahrens (25)
Delft University of Technology, The Netherlands; University of Birmingham, United Kingdom

Takahito Aoto (31)
Niigata University, Japan

Victor Arrial (1)
Université Paris Cité - CNRS - IRIF, France

Franz Baader (16)
Theoretical Computer Science, TU Dresden, Germany; SCADS.AI Dresden/Leipzig, Germany

Patrick Baillot (12)
Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

Jialu Bao (17)
Cornell University, Ithaca, NY, USA

Małgorzata Biernacka (11)
Institute of Computer Science, University of Wrocław, Poland

Dariusz Biernacki (11)
Institute of Computer Science, University of Wrocław, Poland

Filip Cano (2)
Graz University of Technology, Austria

Evan Cavallo (22)
Department of Computer Science and Engineering, University of Gothenburg, Sweden

Şerban-Ion Cercelescu (28)
Department of Computer Science, University of Oxford, Oxford, UK

Camil Champin (6)
École Normale Supérieure de Lyon, France

Kostia Chardonnet (19)
Department of Computer Science and Engineering, University of Bologna, Italy

Carmen-Maria Constantin (28)
Department of Computer Science, University College London, London, UK

Nathan Corbyn (30)
University of Oxford, Oxford, UK

Mateus de Oliveira Oliveira (33)
Department of Computer and Systems Sciences, Stockholm University, Sweden; Department of Informatics, University of Bergen, Norway

Pablo Donato (5)
LIX, École Polytechnique, Palaiseau, France

Maximilian Doré (22)
Department of Computer Science, University of Oxford, United Kingdom

Andrej Dudenhefner (8)
TU Dortmund University, Germany

Aloÿs Dufour (29)
Université Sorbonne Paris Nord, LIPN, CNRS, Villetaneuse, France

Thiago Felicissimo (21)
Université Paris-Saclay, INRIA project Deducteam, Laboratoire Méthodes Formelles, ENS Paris-Saclay, France

Jürgen Giesl (16)
RWTH Aachen University, Aachen, Germany

Tao Gu (17)
University College London, UK

Giulio Guerrieri (1)
University of Sussex, Department of Informatics, Brighton, United Kingdom

Benoît Guillemet (7)
École normale supérieure Paris-Saclay, France

Lukas Heidemann (30)
University of Oxford, Oxford, UK

Thomas A. Henzinger (2)
Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Hugo Herbelin (26)
Université de Paris Cité, Inria, CNRS, IRIF, France

Nao Hirokawa (13)
JAIST, Nomi, Japan

9th International Conference on Formal Structures for Computation and Deduction (FSCD 2024).
Editor: Jakob Rehof
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Justin Hsu (17)
Cornell University, Ithaca, NY, USA

Nick Hu (30)
University of Oxford, Oxford, UK

Sohei Ito (18)
Nagasaki University, Nagasaki, Japan

Junyoung Jang (15)
McGill University, Montreal, Canada

Ambrus Kaposi (10)
Eötvös Loránd University, Budapest, Hungary

Georgios Alexandros Kavvos (14)
University of Bristol, United Kingdom

Marie Kerjean (9)
CNRS, Université Sorbonne Paris Nord, France

Delia Kesner (1)
Université Paris Cité - CNRS - IRIF, France

Jad Koleilat (26)
Université Paris Cité, France

Konstantin Kueffner (2)
Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Bettina Könighofer (2)
Graz University of Technology, Austria

Adrienne Lancelot (23)
Inria & LIX, Ecole Polytechnique, UMR 7161, France; Université Paris Cité, CNRS, IRIF, Paris, France

Jean-Simon Pacaud Lemay (9)
School of Mathematical and Physical Sciences, Macquarie University, Sydney, Australia

Louis Lemonnier (19)
Université Paris-Saclay, CNRS, ENS Paris-Saclay, Inria, Laboratoire Méthodes Formelles, Gif-sur-Yvette, France

Sergueï Lenglet (11)
Université de Lorraine, Nancy, France; Université Sorbonne Paris Nord, France

Salvador Lucas (32)
DSIC & VRAIN, Universitat Politècnica de València, Spain, Spain

Assia Mahboubi (7)
Nantes Université, École Centrale Nantes, CNRS, INRIA, LS2N, UMR 6004, France; Vrije Universiteit Amsterdam, The Netherlands

Kaushik Mallik (2)
Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Ralph Matthes (25)
IRIT, Université de Toulouse, CNRS, Toulouse INP, UT3, Toulouse, France

Damiano Mazza (29)
CNRS, LIPN, Université Sorbonne Paris Nord, Villetaneuse, France

Samuel Mimram (6)
LIX, CNRS, École polytechnique, Institut Polytechnique de Paris, Palaiseau, France

Anders Mörtberg (22)
Department of Mathematics, Stockholm University, Sweden

Naoki Nishida (31)
Nagoya University, Japan

Émile Oleon (6)
LIX, CNRS, École polytechnique, Institut Polytechnique de Paris, Palaiseau, France

Daniele Pautasso (8)
University of Turin, Italy

Frank Pfenning (15)
Carnegie Mellon University, Pittsburgh, USA

Robin Piedeleu (20)
University College London, United Kingdom

Brigitte Pientka (15)
McGill University, Montreal, Canada

Matthieu Piquerez (7)
Nantes Université, École Centrale Nantes, CNRS, INRIA, LS2N, UMR 6004, France

Sophia Roshal (15)
Carnegie Mellon University, Pittsburgh, USA

Claudio Sacerdoti Coen (24)
Alma Mater Studiorum - Università di Bologna, Italy

Teppei Saito (13)
JAIST, Nomi, Japan

Victor Sannier (12)
Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

Chiara Sarti (30)
University of Cambridge, Cambridge, UK

Alan Schmitt (11)
INRIA, France

Jonas Schöpf (31)
University of Innsbruck, Austria

Alexandra Silva (17, 20)
Cornell University, Ithaca, NY, USA

Calin Tataru (30)
University of Cambridge, Cambridge, UK

Makoto Tatsuta (18)
National Institute of Informatics / Sokendai, Tokyo, Japan

René Thiemann (27)
University of Innsbruck, Austria

Mateo Torres-Ruiz (20)
University College London, United Kingdom

Sebastian Ullrich (3)
Lean Focused Research Organization, USA

Farhad Vadiee (33)
Department of Informatics, University of Bergen, Norway

Benoît Valiron (19)
Université Paris-Saclay, CNRS, CentraleSupélec, ENS Paris-Saclay, Inria, Laboratoire Méthodes Formelles, Gif-sur-Yvette, France

Niels van der Weide (4)
Institute for Computing and Information Sciences, Radboud University, Nijmegen, The Netherlands

Jamie Vicary (30)
University of Cambridge, Cambridge, UK

Théo Winterhalter (21)
Université Paris-Saclay, INRIA project Deducteam, Laboratoire Méthodes Formelles, ENS Paris-Saclay, France

Kobe Wullaert (25)
Delft University of Technology, The Netherlands

Szumi Xie (10)
Eötvös Loránd University, Budapest, Hungary

Akihisa Yamada (27)
National Institute of Advanced Industrial Science and Technology, Tokyo, Japan

Fabio Zanasi (17, 20)
University College London, UK; University of Bologna, OLAS team (INRIA), Italy

Meaningfulness and Genericity in a Subsuming Framework

Delia Kesner
Université Paris Cité – CNRS – IRIF, France

Victor Arrial
Université Paris Cité – CNRS – IRIF, France

Giulio Guerrieri
University of Sussex, Department of Informatics, Brighton, United Kingdom

Abstract

This paper studies the notion of meaningfulness for a unifying framework called `dBang`-calculus, which subsumes both call-by-name (`dCBN`) and call-by-value (`dCBV`). We first define meaningfulness in `dBang` and then characterize it by means of typability and inhabitation in an associated non-idempotent intersection type system previously appearing in the literature. We validate the proposed notion of meaningfulness by showing two properties: (1) consistency of the smallest theory, called $\mathcal{H}$, equating all meaningless terms, and (2) genericity, stating that meaningless subterms have no bearing on the significance of meaningful terms. The theory $\mathcal{H}$ is also shown to have a unique consistent and maximal extension $\mathcal{H}^*$, which coincides with a well-known notion of observational equivalence. Last but not least, we show that the notions of meaningfulness and genericity in the literature for `dCBN` and `dCBV` are subsumed by the corresponding ones proposed here for the `dBang`-calculus.

2012 ACM Subject Classification Theory of computation → Operational semantics

Keywords and phrases Lambda calculus, Solvability, Meaningfulness, Inhabitation, Genericity

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.1

Category Invited Talk

Related Version *Full Version*: `https://arxiv.org/abs/2404.06361` [50]

1 Introduction

A common line of research in logic and theoretical computer science is to find unifying frameworks that subsume different paradigms, systems or calculi. Examples are call-by-push-value [54, 55], polarized system LU [45], linear calculi [57, 58, 72], bang-calculus [38, 39, 23, 24], system L [62, 35], ecumenical systems [68], monadic calculus [60, 61], and others [71, 40, 73].

The relevance of these unifying frameworks lies in the range of properties and models they encompass. Finding *unifying and simple primitives, tools and techniques* to reason about properties of different systems is challenging, and provides a deeper and more abstract understanding of these properties. The advantages of this kind of approach are numerous, for instance the *several-for-one deal*: study a property in a unifying framework gives appropriate intuitions and hints for free for all the subsumed systems. The aim of this paper is to go beyond the state of the art in a framework subsuming the *call-by-name* and *call-by-value* evaluation mechanisms, by unifying their notions of *meaningful* (and *meaningless*) programs.

Call-by-name and call-by-value. Every programming language implements a particular evaluation strategy, specifying when and how parameters are evaluated during function calls. For example, in call-by-value (`CBV`), the argument is evaluated before being passed to the function, while in call-by-name (`CBN`) the argument is passed immediately to the function

body, so that it may never be evaluated, or may be re-evaluated several times. These models of computation serve as the basis for many theoretical and practical studies in programming languages and proof assistants, such as OCaml, Haskell, Coq, Isabelle, etc.

The CBN strategy has garnered significant attention in the literature on theoretical studies and is generally perceived as well-established. In contrast, the CBV strategy has received limited attention. Despite their similarities, CBN and CBV strategies have predominantly been studied independently, leading to a fragmented research. This approach not only duplicates research efforts – once for CBN and once again for CBV – but also generally results in ad-hoc methods for dealing with the CBV case that are naively adapted from the CBN one.

Understanding the (logical) duality between CBN and CBV (*e.g.* [34]) marked a significant step towards properly unifying these models. It paved the way for the emergence of Call-by-Push-Value (CBPV), a unifying framework introduced by P.B. Levy [54, 55] which *subsumes*, among others, CBN and CBV denotational and operational semantics thanks to the distinction between *computations* and *values*, according to the slogan "a value is, a computation does". This framework attracts growing attention: proving advanced properties of a single *unifying paradigm*, and subsequently instantiate them for a *wide range* of computational models.

The distant Bang-calculus. Drawing inspiration from Girard's Linear Logic (LL) [44] and the interpretation of CBPV into LL [38], Ehrhard and Guerrieri [39] introduced an (untyped) restriction of CBPV, named Bang-*calculus*, already capable of subsuming both CBN and CBV. It is obtained by enriching the λ-calculus with two modalities ! and its dual der. The modality ! actually plays a twofold role: it freezes the evaluation of subterms (called *thunk* in CBPV), and it marks what can be duplicated or erased during evaluation (*i.e.* copied an arbitrary number of times, including zero). The modality der annihilates the effect of !, effectively restoring computation and eliminating duplicability. Embedding CBN or CBV into the Bang-calculus via *Girard's translations* simply consists in decorating λ-terms with ! and der, thereby forcing one model of computation or the other one. Thanks to these elementary modalities and embeddings, the Bang-calculus eases the identification of shared behaviors and properties of CBN and CBV, encompassing both syntactic and semantic aspects of them.

The original Bang-calculus [39] uses some permutation rules, similar to the ones used in [70, 30], that unveil hidden redexes and unblock reductions that otherwise would be stuck. These permutation rules make the calculus *adequate*, preventing some normal forms from being observationally equivalent to non-terminating terms. A major drawback is that the resulting combined reduction is not confluent (Page 6 in [39]). The *distant* Bang-*calculus* (dBang) [23, 24] was proposed as an adequate *and* confluent alternative. This is achieved by enriching the syntax with *explicit substitutions*, in the vein of Accattoli and Kesner's linear substitution calculus [7, 9, 1, 2] (generalizing in turn Milner's calculus [59, 51]), thanks to rewrite rules that act *at a distance*, so that permutation rules are no longer needed.

In this paper, we focus on dBang, and its relations with dCBN [9, 1] and dCBV [11], which are *distant* adequate variants of the CBN and CBV λ-calculi. This unifying framework is fruitful, subsuming numerous dCBN and dCBV properties through their associated embedding, as for instance big step semantics: evaluating the result from the dCBN/dCBV embedding of a given program t with the dBang model actually corresponds to the embedding of the result of evaluating the original program t with the dCBN/dCBV model. In other words, dBang is a language that breaks down the dCBN and dCBV paradigms into elementary primitives.

Let us now review the state of the art by discussing some advanced properties of programming languages that have been studied in the literature by using the unifying approach dBang. Some of these results, including this work, strongly rely on semantical tools such as quantitative types. To ensure clarity regarding the state of the art, let us briefly discuss in first place the main ideas behind quantitative types.

Quantitative Type Systems. *Intersection type systems* [31, 32] increase the typability power on λ-terms with respect to simple types by introducing a new *intersection* type constructor $\wedge$ that is associative, commutative and *idempotent* (*i.e.* $\sigma \wedge \sigma = \sigma$). Intersection types allow terms to have different types simultaneously, *e.g.* a term has type $\sigma \wedge \tau$ whenever it has both types σ and τ. They constitute a powerful tool to reason about *qualitative* properties of programs. For example, different notions of normalization can be characterized using intersection types [67, 33], in that a term t is typable in a given system if and only if t is normalizing (as a consequence, typability in these systems is undecidable). An alternative version of intersection type systems for the λ-calculus, called *non-idempotent* [43, 36], is obtained by dropping idempotence. In such a setting, a term of type $\sigma \wedge \sigma \wedge \tau$ can be seen as a resource used exactly once as a data of type τ and twice as a data of type σ. Interestingly, such type systems provide not only qualitative characterizations of different operational properties, but also *quantitative* ones: *e.g.* a term t is still typable if and only if t is normalizing, moreover any type derivation of t gives an *upper bound* to the execution time for t (the number of steps to reach a normal form) [37]. These upper bounds can be further refined into *exact measure* using *tight non-idempotent typing systems*, as pioneered in [4].

State of the Art. This paper contributes to a broader initiative aimed at consolidating the theory of `dCBN` and `dCBV`, by unifying them into `dBang`. Several results have already been factorized and generalized in this framework, we now revisit some of them.

In [46], it is shown that the interpretation of a term t in any denotational model of `CBN`/`CBV` obtained from LL is included in the interpretation of the `CBN`/`CBV` translation of t in any denotational model of `Bang` obtained from LL. The reverse inclusion also holds for `CBN` but not for `CBV`. In particular, these results apply to *typability* in non-idempotent intersection type systems inspired by LL. Indeed, typing is preserved by Girard's translations, meaning that if a term is typable in the `CBN`/`CBV` type system, then its `CBN`/`CBV` translation is typable in the type system $\mathcal{B}$ for `Bang`, using the same types. The converse holds for `CBN` but not for `CBV`. In [23, 24], the `CBV` typing system is modified so that the reverse implication also holds. Moreover, an extension of Girard's `CBN` translation to `dCBN` and a *new* `CBV` translation for `dCBV` are proposed. Similar typing preservation results have been obtained in [52] for the translations in [23, 24], but for the more precise notion of tight typing introduced in [4].

Retrieving *dynamic* properties from `Bang` into `CBN` and `CBV` turns out to be a more intricate task, especially in their *adequate* (distant) variant [23, 41, 24].

In [46] it is shown that `CBN` and `CBV` can be simulated by *reduction* in `Bang` through Girard's original translations. But the `CBV` translation fails to preserve *normal forms*, as some `CBV` normal forms translate to reducible terms in `Bang`. This issue is solved in `dBang` [23, 24], thanks to the *new* `CBV` translation for `dCBV` previously mentioned. In the end, reductions *and* normal forms are preserved by both the `CBN` and the new `CBV` translations.

Even if `dCBN` and `dCBV` can be both simulated by *reduction* in `dBang`, the converse, known as *reverse simulation*, holds for `dCBN` but fails for `dCBV` [24, 14]: a `dBang` reduction sequence from a term in the image of the `dCBV` embedding may not correspond to a valid reduction sequence in `dCBV`. Yet another new `dCBV` translation is proposed in [14] so that simulation and reverse simulation are now recovered.

Another major contribution concerns the *inhabitation* problem: given an environment Γ (a type assignment for variables) and a type σ, decide whether there is a term t that can be typed with σ under the environment Γ. While inhabitation was shown [74] to be *undecidable* in `CBN` for idempotent intersection type systems, it turns out to be *decidable* [25, 28] in the non-idempotent setting. Decidability of the inhabitation problem leads to the development

of automatic tools for type-based *program synthesis* [56, 21], whose goal is to construct a program – the term t – that satisfies some high-level formal specification, expressed as a type σ with some assumptions described by the environment Γ. It has been proved in [13] that the algorithms deciding the inhabitation problem for $\mathtt{dCBN}$ and $\mathtt{dCBV}$ can be inferred from the corresponding one for $\mathtt{dBang}$, thus providing a unified solution to this relevant problem.

Meaningfulness and Genericity. In this work, we aim to unify the notions of meaningfulness and genericity in $\mathtt{dCBN}$ and $\mathtt{dCBV}$ so as to derive them from the respective ones in $\mathtt{dBang}$.

A naive approach to set a semantics for the pure untyped λ-calculus is to define the meaning of a β-normalizing λ-term as its normal form, and equating all λ-terms that do not β-normalize. The underlying idea is that, as β-reduction represents evaluation and a normal form stands for its outcome, all non-β-normalizing λ-terms (*i.e.* diverging programs) are then considered as meaningless. However, this simplistic approach is flawed, as thoroughly discussed in [20]. For example, any λ-theory equating all non-β-normalizing λ-terms is inherently inconsistent – it effectively equates all λ-terms, not just the meaningless ones!

Alternatively, during the 70s, Wadsworth [75, 76] and Barendregt [17, 18, 19, 20] showed that the meaningful ($\mathtt{CBN}$) λ-terms can be identified with the *solvable* ones. Solvability is defined in a rather technical way: a λ-term t is *solvable* if there is a special kind of context, called *head* context $\mathtt{H}$, sending t to the identity function $\mathtt{I} = \lambda z.z$, meaning that $\mathtt{H}\langle t\rangle$ β-reduces to $\mathtt{I}$. Roughly, a solvable λ-term t may be divergent, but its diverging subterms can be eliminated by supplying the right arguments to t via an appropriate interaction with a suitable head context $\mathtt{H}$. For instance, in $\mathtt{CBN}$, $x\Omega$ is divergent but solvable using the head context $\mathtt{H} = (\lambda x.\diamond)(\lambda y.\mathtt{I})$. It turns out that *unsolvable* λ-terms constitutes a strict subset of the non-β-normalizing ones. Moreover, the smallest λ-theory that equates all unsolvable λ-terms is *consistent* (*i.e.* it does not equate all terms). In Barendregt's book [20], these results rely on a keystone property known as (*full*) *genericity*, which states that meaningless subterms are computationally irrelevant – in the sense that they do not play any role – in the evaluation of β-normalizing terms. Formally, if t is *unsolvable* and $\mathtt{C}\langle t\rangle$ β-reduces to *some* β-normal term u for some context $\mathtt{C}$, then $\mathtt{C}\langle s\rangle$ β-reduces to u for *every* λ-term s. This property stands as a fool guard that the choice of meaningfulness is adequate. A variant of genericity [16], called *surface* in [15] and *light* in [10], states that any meaningless subterm t is irrelevant in a meaningful term $\mathtt{C}\langle t\rangle$ in that $\mathtt{C}\langle s\rangle$ is still meaningful, for every term u.

Meaningfulness was also studied for first order rewriting systems [48] and other strategies of the λ-calculus [71]. Notably, finding the correct notion of meaningfulness for $\mathtt{CBV}$ has been a challenge [5, 6, 15]. Similarly, an extension of the $\mathtt{dCBN}$ was studied [29, 26] in the framework of a λ-calculus equipped with pattern matching for pairs. The use of different data structures in the language – functions and pairs – makes meaningfulness more challenging. Indeed, it was shown that meaningfulness cannot be characterized only by means of typability alone, as in $\mathtt{CBN}$ and $\mathtt{CBV}$, but also requires some additional conditions stated in terms of the inhabitation problem previously mentioned. This result for the λ-calculus with patterns inspired the characterization of meaningfulness for $\mathtt{dBang}$ that we provide in this paper. Genericity for $\mathtt{dCBN}$ and the more subtle case of $\mathtt{dCBV}$ was recently proved in [15].

Our Contributions. We first define meaningfulness for $\mathtt{dBang}$, for which we provide a characterization by means of typability *and* inhabitation. As a second contribution, we validate this notion of meaningfulness twofold: meaningless terms enjoy surface genericity, and the smallest $\lambda_{\mathtt{dBang}}$-theory $\mathcal{H}_{\mathtt{dBang}}$ obtained by equating all the meaningless terms is consistent. Moreover, we show that $\mathcal{H}_{\mathtt{dBang}}$ admits a unique maximal consistent extension

$\mathcal{H}^*_{\mathtt{dBang}}$ and show that it coincides with the well-known notion of observational equivalence. Last but not least, as a third contribution, we show that the notions of meaningfulness in the literature for dCBN and dCBV are subsumed by the one proposed here for dBang. We also obtain surface genericity for dCBN and dCBV as a consequence of the genericity property for dBang, and relate the theories $\mathcal{H}_{\mathtt{dBang}}$ and $\mathcal{H}^*_{\mathtt{dBang}}$ (in dBang) to the corresponding ones in dCBN and dCBV. Detailed proofs of our results can be found in [50].

Roadmap. Section 2 recalls dBang and its quantitative type system $\mathcal{B}$. Section 3 defines meaningfulness for dBang, and characterizes it in terms of typability and inhabitation in the type system $\mathcal{B}$. Section 4 addresses surface genericity and the construction of the theories $\mathcal{H}_{\mathtt{dBang}}$ and $\mathcal{H}^*_{\mathtt{dBang}}$, while Section 5 establishes a precise relationship between meaningless and genericity in dCBN/dCBV and their corresponding notions in dBang. Section 6 discusses future and related work and concludes.

2 The dBang-Calculus

2.1 Syntax and Operational Semantics

We introduce the syntax of the *distant* Bang-*calculus* (dBang) [23, 24]. Given a countably infinite set $\mathcal{X}$ of variables $x, y, z, \dots$, the set $\Lambda_!$ of *terms* is inductively defined as follows:

$$\textbf{(Terms)} \quad t, u, s \ := \ x \in \mathcal{X} \mid t\,u \mid \lambda x.t \mid t[x\backslash u] \mid !t \mid \mathtt{der}(t)$$

The set $\Lambda_!$ includes **variables** x, **abstractions** $\lambda x.t$ and **applications** tu (as in the λ-calculus), and three other constructors: a **closure** $t[x\backslash u]$ representing a pending **explicit substitution (ES)** $[x\backslash u]$ on a term t, a **bang** $!t$ to freeze the execution of t, and a **dereliction** $\mathtt{der}(t)$ to fire again the frozen term t. The **argument** of an application $t\,u$ (resp. a closure $t[x\backslash u]$) is the subterm u. From now on, we set $\mathtt{I}_! := \lambda z.!z$, $\Delta_! := \lambda x.x!x$, and $\Omega_! := \Delta_!!\Delta_!$.

Abstractions $\lambda x.t$ and closures $t[x\backslash u]$ bind the variable x in the term t. **Free** and **bound** variables are defined as expected, in particular $\mathtt{fv}(\lambda x.t) := \mathtt{fv}(t) \setminus \{x\}$ and $\mathtt{fv}(t[x\backslash u]) := \mathtt{fv}(u) \cup (\mathtt{fv}(t) \setminus \{x\})$. The usual notion of α-conversion [20] is extended to $\Lambda_!$, and terms are identified up to α-conversion. We denote by $t\{x\backslash u\}$ the usual (capture avoiding) meta-level substitution of the term u for all free occurrences of the variable x in the term t.

List contexts (L), **surface contexts** (S) and **full contexts** (F), which can be seen as terms containing exactly one **hole** $\diamond$, are inductively defined as follows:

$$\begin{array}{rrcl}
\textbf{(List Contexts)} & \mathtt{L} & ::= & \diamond \mid \mathtt{L}[x\backslash t] \\
\textbf{(Surface Contexts)} & \mathtt{S} & ::= & \diamond \mid \mathtt{S}\,t \mid t\,\mathtt{S} \mid \lambda x.\mathtt{S} \mid \mathtt{der}(\mathtt{S}) \mid \mathtt{S}[x\backslash t] \mid t[x\backslash \mathtt{S}] \\
\textbf{(Full Contexts)} & \mathtt{F} & ::= & \diamond \mid \mathtt{F}\,t \mid t\,\mathtt{F} \mid \lambda x.\mathtt{F} \mid \mathtt{der}(\mathtt{F}) \mid \mathtt{F}[x\backslash t] \mid t[x\backslash \mathtt{F}] \mid !\mathtt{F}
\end{array}$$

List and surface contexts are special cases of full contexts. The hole can occur everywhere in full contexts, while it is forbidden under ! in surface contexts. For example, $y\,(\lambda x.\diamond)$ is a surface context hence a full context, while $(!\diamond)[x\backslash \mathtt{I}_!]$ is a full context but not a surface one. We write $\mathtt{F}\langle t\rangle$ for the term obtained by replacing the hole in F with the term t.

The following **rewrite rules** are the base components of the reduction system of dBang. Any term having the shape of the left-hand side of one of these three rules is called a **redex**.

$$\mathtt{L}\langle\lambda x.t\rangle\,u \mapsto_{\mathtt{dB}} \mathtt{L}\langle t[x\backslash u]\rangle \qquad t[x\backslash \mathtt{L}\langle !u\rangle] \mapsto_{\mathtt{s!}} \mathtt{L}\langle t\{x\backslash u\}\rangle \qquad \mathtt{der}(\mathtt{L}\langle !t\rangle) \mapsto_{\mathtt{d!}} \mathtt{L}\langle t\rangle$$

Rule dB (resp. s!) is assumed to be capture free: no free variable of u (resp. t) is captured by the list context L. The rule dB fires a β-redex and generates an ES. The rule s! operates a substitution provided its argument is a bang: only bang terms can be erased or duplicated,

and they lose their bang when the substitution is performed. The rule $\mathtt{d!}$ opens a bang. All these rewrite rules act *at a distance* [7, 9, 2]: the main constructors involved in the rule can be separated by a finite – possibly empty – list context $\mathtt{L}$ of ES. This mechanism unblocks redexes that would otherwise be stuck, *e.g.* $(\lambda x.x)[y\backslash w]!z \mapsto_{\mathtt{dB}} x[x\backslash !z][y\backslash w]$ fires a β-redex where $\mathtt{L} = \diamond[y\backslash w]$ is the list context in between the abstraction $\lambda x.x$ and the argument $!z$.

The **surface reduction** $\to_{\mathrm{S}}$ is the surface closure of the three rewrite rules $\mathtt{dB}$, $\mathtt{s!}$ and $\mathtt{d!}$, *i.e.* $\to_{\mathrm{S}}$ only fires redexes in surface contexts (not under bang). Similarly, the **full reduction** $\to_{\mathrm{F}}$ is the full closure of the three rewrite rules $\mathtt{dB}$, $\mathtt{s!}$ and $\mathtt{d!}$, *i.e.* $\to_{\mathrm{F}}$ fires redexes in any full contexts and thus the bang loses its freezing behavior. For example,

$$(\lambda x.!\mathtt{der}(!x))!y \;\to_{\mathrm{S}}\; (!\mathtt{der}(!x))[x\backslash !y] \;\to_{\mathrm{S}}\; !(\mathtt{der}(!y)) \;\to_{\mathrm{F}}\; !y$$

The first two $\to_{\mathrm{S}}$-steps are $\to_{\mathrm{F}}$-steps too, the last one is not a $\to_{\mathrm{S}}$-step. We denote by $\to_{\mathrm{S}}^*$ the reflexive-transitive closure of $\to_{\mathrm{S}}$, and similarly for $\to_{\mathrm{F}}$. A reduction $\to_{\mathcal{R}}$ is **confluent** if for all t, u_1, u_2 such that $t \to_{\mathcal{R}}^* u_1$ and $t \to_{\mathcal{R}}^* u_2$, there is s such that $u_1 \to_{\mathcal{R}}^* s$ and $u_2 \to_{\mathcal{R}}^* s$.

▶ **Theorem 1.** *The reductions $\to_{\mathrm{S}}$ and $\to_{\mathrm{F}}$ are confluent.*

Proof. For $\to_{\mathrm{S}}$ see [23], for $\to_{\mathrm{F}}$ see [50]. ◀

A term t is a **surface** (resp. **full**) **normal form** if there is no u such that $t \to_{\mathrm{S}} u$ (resp. $t \to_{\mathrm{F}} u$). A term t is **surface** (resp. **full**) **normalizing** if $t \to_{\mathrm{S}}^* u$ (resp. $t \to_{\mathrm{F}}^* u$) for some surface (resp. full) normal form u. Since $\to_{\mathrm{S}} \subsetneq \to_{\mathrm{F}}$, some terms may be surface-normalizing but not full-normalizing, *e.g.* $\lambda x.!(\mathtt{der}(!\Omega_!))$.

As a matter of fact, some ill-formed terms are not redexes but neither represent a desired computation result. They are called **clashes** and have one of the following forms:

$$\mathtt{L}\langle !t\rangle\, u \qquad t[x\backslash \mathtt{L}\langle \lambda x.u\rangle] \qquad \mathtt{der}(\mathtt{L}\langle \lambda x.t\rangle) \qquad t(\mathtt{L}\langle \lambda x.u\rangle) \text{ if } t \neq \mathtt{L}'\langle \lambda y.s\rangle$$

This *static* notion of clash is lifted to a *dynamic* level. A term t is **surface** (resp. **full**) **clash-free** if it does not surface (resp. full) reduce to a term with a clash in surface (resp. full) position, *i.e.* if there are no surface (resp. full) context $\mathtt{S}$ (resp. $\mathtt{F}$) and clash c such that $t \to_{\mathrm{S}}^* \mathtt{S}\langle c\rangle$ (resp. $t \to_{\mathrm{F}}^* \mathtt{F}\langle c\rangle$). For example, $x!(y(\lambda z.z))$ is surface clash-free but not full clash-free as it has a clash $y(\lambda z.z)$ under a bang. Both notions are stable under reduction.

Finally, some terms contain neither redexes nor clashes. A **surface** (resp. **full**) **clash-free normal form** is a surface (resp. full) normal form which is also surface (resp. full) clash-free, as *e.g.* the term xx. These are the results of the computation, and they can even be *syntactically* characterized by the grammar $\mathtt{no}_{\mathrm{S}}$ below.

$$\mathtt{ne}_{\mathrm{S}} \coloneqq x \in \mathcal{X} \mid \mathtt{ne}_{\mathrm{S}}\,\mathtt{na}_{\mathrm{S}} \mid \mathtt{der}(\mathtt{ne}_{\mathrm{S}}) \mid \mathtt{ne}_{\mathrm{S}}[x\backslash \mathtt{ne}_{\mathrm{S}}] \qquad \mathtt{na}_{\mathrm{S}} \coloneqq\ !t \mid \mathtt{ne}_{\mathrm{S}} \mid \mathtt{na}_{\mathrm{S}}[x\backslash \mathtt{ne}_{\mathrm{S}}]$$
$$\mathtt{nb}_{\mathrm{S}} \coloneqq \mathtt{ne}_{\mathrm{S}} \mid \lambda x.\mathtt{no}_{\mathrm{S}} \mid \mathtt{nb}_{\mathrm{S}}[x\backslash \mathtt{ne}_{\mathrm{S}}] \qquad \mathtt{no}_{\mathrm{S}} \coloneqq \mathtt{na}_{\mathrm{S}} \mid \mathtt{nb}_{\mathrm{S}}$$

▶ **Lemma 2** ([23]). *Let $t \in \Lambda_!$, then $t \in \mathtt{no}_{\mathrm{S}}$ iff t is a surface clash-free normal form.*

2.2 Quantitative Typing System

We present the quantitative typing system $\mathcal{B}$ [23], based on [43, 36], for $\mathtt{dBang}$. It contains arrow and intersection types. Intersections are associative, commutative but *not idempotent*, thus an intersection type is represented by a (possibly empty) *finite multiset* $[\sigma_i]_{i\in I}$. Given a countably infinite set $\mathcal{TV}$ of type variables $\alpha, \beta, \gamma, \ldots$, we define by mutual induction:

$$\begin{array}{rrcl} (\textbf{Types}) & \sigma, \tau, \rho & \coloneqq & \alpha \in \mathcal{TV} \mid \mathcal{M} \mid \mathcal{M} \Rightarrow \sigma \\ (\textbf{Multitypes}) & \mathcal{M}, \mathcal{N} & \coloneqq & [\sigma_i]_{i\in I} \text{ where } I \text{ is a finite set} \end{array}$$

$$\frac{}{x : [\sigma] \vdash x : \sigma}\ (var) \qquad \frac{\Gamma \vdash t : \mathcal{M} \Rightarrow \sigma \qquad \Delta \vdash u : \mathcal{M}}{\Gamma + \Delta \vdash t\, u : \sigma}\ (app) \qquad \frac{(\Gamma_i \vdash t : \sigma_i)_{i\in I} \quad I \text{ finite}}{+_{i\in I}\Gamma_i \vdash\, !t : [\sigma_i]_{i\in I}}\ (bg)$$

$$\frac{\Gamma, x : \mathcal{M} \vdash t : \sigma}{\Gamma \vdash \lambda x.t : \mathcal{M} \Rightarrow \sigma}\ (abs) \qquad \frac{\Gamma, x : \mathcal{M} \vdash t : \sigma \qquad \Delta \vdash u : \mathcal{M}}{\Gamma + \Delta \vdash t[x\backslash u] : \sigma}\ (es) \qquad \frac{\Gamma \vdash t : [\sigma]}{\Gamma \vdash \mathtt{der}(t) : \sigma}\ (der)$$

Figure 1 Type System $\mathcal{B}$ for the dBang-calculus.

A (**type**) **environment**, noted Γ or Δ, is a function from variables to multitypes, assigning the **empty multitype** $[\,]$ to all variables except a finite number (possibly zero). The **empty environment**, noted $\emptyset$, maps every variable to $[\,]$. The **domain** of Γ is $\mathtt{dom}(\Gamma) = \{x \in \mathcal{X} \mid \Gamma(x) \neq [\,]\}$, the **image** of Γ is $\mathtt{im}(\Gamma) = \{\Gamma(x) \mid x \in \mathtt{dom}(\Gamma)\}$. Given the environments Γ and Δ, $\Gamma + \Delta$ is the environment mapping x to $\Gamma(x) \uplus \Delta(x)$, where $\uplus$ denotes multiset union; and $+_{i\in I}\Delta_i$ (with I finite) is its n-ary extension, in particular $+_{i\in I}\Delta_i = \emptyset$ if $I = \emptyset$. An environment Γ is denoted by $x_1 : \mathcal{M}_1, \dots, x_n : \mathcal{M}_n$ when the x_i's are pairwise distinct variables and $\Gamma(x_i) = \mathcal{M}_i$ for all $1 \leq i \leq n$, and $\Gamma(y) = [\,]$ for $y \notin \{x_1, \dots, x_n\}$.

A **typing** is a pair $(\Gamma; \sigma)$, where Γ is an environment and σ is a type. A (**typing**) **judgment** is a tuple of the form $\Gamma \vdash t : \sigma$, where $(\Gamma; \sigma)$ is a typing and t is a term (the **subject** of the judgment). The typing system $\mathcal{B}$ for dBang is defined by the rules in Figure 1. The axiom rule (*var*) is relevant, *i.e.* there is no weakening. Rules (*abs*), (*app*) and (*es*) are standard. Rule (*bg*) has as many premises as elements in the finite (possibly empty) index set I, and its conclusion types $!t$ with a multitype *gathering* all the (possibly different) types in the premises typing t. In particular, when $I = \emptyset$, the rule has no premises, and it types *any* term $!t$ with $[\,]$, leaving the *subterm* t *untyped*. Rule (*der*) forces the argument of a dereliction to be typed by a multitype of cardinality 1.

A (**type**) **derivation** in system $\mathcal{B}$ is a tree obtained by applying the rules in Figure 1. The judgment at the root of the type derivation Π is the **conclusion** of Π. We write $\Pi \triangleright_{\mathcal{B}} \Gamma \vdash t : \sigma$ when Π is a derivation in system $\mathcal{B}$ with conclusion $\Gamma \vdash t : \sigma$, and $\triangleright_{\mathcal{B}} \Gamma \vdash t : \sigma$ if there exists some derivation $\Pi \triangleright_{\mathcal{B}} \Gamma \vdash t : \sigma$. A term t is **$\mathcal{B}$-typable** if $\triangleright_{\mathcal{B}} \Gamma \vdash t : \sigma$ for some typing $(\Gamma; \sigma)$.

System $\mathcal{B}$ enjoys subject reduction and expansion with respect to $\to_{\mathrm{F}}$, and characterizes surface-normalizing clash-free terms.

▶ **Theorem 3** ([23, 13]). *Let $t, u \in \Lambda_!$.*

1. *If $t \to_{\mathrm{F}} u$, then for any typing $(\Gamma; \sigma)$, one has $\triangleright_{\mathcal{B}} \Gamma \vdash t : \sigma$ if and only if $\triangleright_{\mathcal{B}} \Gamma \vdash u : \sigma$.*
2. *t is $\mathcal{B}$-typable if and only if t surface-reduces to a surface clash-free normal form.*

3 Meaningfulness = Typability + Inhabitation

In this section, we introduce the notion of meaningfulness for dBang and we establish a logical characterization of meaningfulness via system $\mathcal{B}$. Intuitively, a term t is meaningful if it can be supplied by some arguments (possibly binding some free variables of t) so that it reduces to some observable term. In dBang, the observables are the bang terms since they are the only terms enabling substitution to be fired.

▶ **Definition 4.** *A term t is **dBang-meaningful** if there are a testing context $\mathtt{T}$ and $u \in \Lambda_!$ such that $\mathtt{T}\langle t\rangle \to_{\mathrm{S}}^* !u$, where* testing contexts *are defined by the grammar $\mathtt{T} := \diamond \mid \mathtt{T}s \mid (\lambda x.\mathtt{T})s$.*[1] *A term t is **dBang-meaningless** if it is not dBang-meaningful.*

[1] Thanks to a factorization theorem for dBang [14], in our definition of dBang-meaningfulness $\to_{\mathrm{S}}^*$ can equivalently be replaced by $\to_{\mathrm{F}}^*$. For the same reason, the same remark also applies to Definition 14.

$$\dfrac{\dfrac{}{x : [\mathcal{M} \Rightarrow \sigma] \vdash x : \mathcal{M} \Rightarrow \sigma}\,(var) \qquad \dfrac{}{x : [\mathcal{M}] \vdash x : \mathcal{M}}\,(var)}{x : [\mathcal{M} \Rightarrow \sigma, \mathcal{M}] \vdash xx : \sigma}\,(app)$$

Figure 2 A type derivation of xx in system $\mathcal{B}$.

$$\dfrac{\dfrac{\dfrac{}{x : [\alpha] \vdash x : \alpha}\,(var)}{x : [\alpha] \vdash !x : [\alpha]}\,(bg)}{\emptyset \vdash \lambda x.!x : [\alpha] \Rightarrow [\alpha]}\,(abs)$$

Figure 3 Inhabitation of $[\alpha] \Rightarrow [\alpha]$ in system $\mathcal{B}$.

For example, $\mathtt{I}_!$ is dBang-meaningful, take the testing context $\mathtt{T} = \diamond\,!u$. Both $\Omega_!$ and $x\Omega_!$ are dBang-meaningless: every testing context they are plugged in cannot erase $\Omega_!$, which is not normalizing and does not reduce to a bang term. Note that all testing contexts are surface, and that the hole in a testing context is always in the functional position of an application, in particular if the hole is in the scope of some λ, then this λ must be applied.

Readers familiar with the advanced theory of λ-calculus may wonder about the relevance of our notion of dBang-meaningfulness. In particular, we could have just naively extended the well-known notion of call-by-name *solvability*: a term t is **dBang-solvable** if there are a testing context $\mathtt{T}$ such that $\mathtt{T}\langle t\rangle \to_{\mathrm{S}}^{*} \mathtt{I}_!$. We found at least two reasons to not use dBang-solvability: the first one is that we would lose consistency of the smallest $\lambda_{\mathtt{dBang}}$-theory generated by equating all dBang-unsolvable terms (see discussion after Proposition 8), while the second one is that we would lose genericity (see discussion after Corollary 11).

In an adequate calculus, meaningfulness is usually characterized both operationally (normalizability) and logically (typability): a term is meaningful iff it is normalizing for a suitable subreduction of the calculus iff it is typable in a suitable type system. Surprisingly, these characterizations are subtler in dBang, because the language has two (incompatible) data structures: abstractions (playing the role of functions) and bangs (playing as values).

A natural idea to operationally characterize dBang-meaningfulness would be *normalizability* by *surface reduction*, but this fails, even if we require the obtained surface normal form to be clash-free. For instance, the term xx is dBang-meaningless despite being a surface clash-free normal form. Indeed, for xx to be dBang-meaningful, a testing context $\mathtt{T}$ would need to provide a term u to substitute the variable x, so that $\mathtt{T}\langle xx\rangle$ would eventually reduce to a bang. However, achieving this requires the term u to reduce to both an abstraction and a bang, which is impossible. Hence, dBang-meaningfulness is not only the ability to produce a surface clash-free normal form, but also to transform this result into an observable.

Concerning a logical characterization of dBang-meaningfulness, *typability* is not enough, at least in system $\mathcal{B}$, since it just characterizes surface clash-free normalization (Theorem 3.2). For instance, the dBang-meaningless term xx seen above is typable in system $\mathcal{B}$. Every type derivation of xx has the form of that in Figure 2, which reveals the conflict when assigning to x both an arrow type $\mathcal{M} \Rightarrow \sigma$ (the type of terms eventually reducing to abstractions) and a multitype $\mathcal{M}$ (the type of terms eventually reducing to bangs). The inhabitation problem can be used to detect such conflicts, allowing for a handy characterization of meaningfulness. Indeed, the multitype $[\mathcal{M} \Rightarrow \sigma, \mathcal{M}]$ assigned to the variable x in Figure 2 is not *inhabited*. Other (naive and unsuccessful) alternatives are discussed in Section 6.

While it seems complex to syntactically establish operational conditions such as (not) reducing to abstractions or bangs, this is easily achieved semantically. Indeed, we establish a logical characterization of dBang-meaningfulness based on *typability* and *inhabitation* in system $\mathcal{B}$, similarly to what happens in the λ-calculus with pairs [12, 29, 26]. Intuitively, suppose that a term t is dBang-meaningful, so there is a testing context $\mathtt{T}$ such that $\mathtt{T}\langle t\rangle$

reduces to an observable, *i.e.* a bang, which can be (trivially) typed with the typing $(\emptyset; [\,])$ in system $\mathcal{B}$. By Theorem 3.1, $\mathtt{T}\langle t\rangle$ must also be typable by the same typing $(\emptyset; [\,])$, meaning that t is *typable* by some environment $x_1 : \mathcal{M}_1, \dots, x_n : \mathcal{M}_m$ and some type $\mathcal{N}_1 \Rightarrow \dots \Rightarrow \mathcal{N}_n \Rightarrow [\,]$, where each of the $\mathcal{M}_i$'s and $\mathcal{N}_i$'s is *inhabited*, *i.e.* there is a term with such a type.

A similar argument holds for other type systems and calculi [29, 26] with their own notions of meaningfulness and observable. The point is to identify the set of types $\mathcal{T}_{\mathcal{S}}^{\mathtt{obs}}$ associated with the observables. In any type system $\mathcal{S}$ whose types are those of Section 2.2, given a set of types $\mathcal{T}_{\mathcal{S}}^{\mathtt{obs}}$ for observable terms, the set of **arguments** $\mathtt{args}_{\mathcal{S}}(\sigma)$ of a type σ is the set of multitypes appearing to the left of arrows, until reaching the type of an observable. Formally, if $\sigma \in \mathcal{T}_{\mathcal{S}}^{\mathtt{obs}}$ then $\mathtt{args}_{\mathcal{S}}(\sigma) \coloneqq \emptyset$, *otherwise* $\mathtt{args}_{\mathcal{S}}(\alpha) \coloneqq \emptyset$, $\mathtt{args}_{\mathcal{S}}(\mathcal{M} \Rightarrow \sigma) \coloneqq \{\mathcal{M}\} \cup \mathtt{args}_{\mathcal{S}}(\sigma)$, and $\mathtt{args}_{\mathcal{S}}(\mathcal{M}) = \emptyset$. In system $\mathcal{B}$, we set $\mathcal{T}_{\mathcal{B}}^{\mathtt{obs}} \coloneqq \{\mathcal{M} \mid \mathcal{M} \text{ multitype}\}$, because bang terms – the observables in dBang – can be only typed by multisets. For example, $\mathtt{args}_{\mathcal{B}}([\tau] \Rightarrow (\mathcal{M} \Rightarrow [\alpha])) = \{[\tau], \mathcal{M}\}$. The cases of dCBN and dCBV type systems are discussed in Section 5, this is why our definitions deal with a generic type system $\mathcal{S}$.

▶ **Definition 5.** *Let $\mathcal{S}$ be a type system and $\mathtt{inh}_{\mathcal{S}}(\cdot)$ be a predicate on the types of $\mathcal{S}$. A set S of types is **inhabited**, noted $\mathtt{inh}_{\mathcal{S}}(S)$, if $\mathtt{inh}_{\mathcal{S}}(\sigma)$ for all $\sigma \in S$. We write $\mathtt{inh}_{\mathcal{S}}(\Gamma)$ if $\mathtt{inh}_{\mathcal{S}}(\mathtt{im}(\Gamma))$. A typing $(\Gamma; \sigma)$ or a judgment $\Gamma \vdash t : \sigma$ is $\mathcal{S}$-**testable** if $\mathtt{inh}_{\mathcal{S}}(\Gamma)$ and $\mathtt{inh}_{\mathcal{S}}(\mathtt{args}_{\mathcal{S}}(\sigma))$. A term t is $\mathcal{S}$-**testable** if $\triangleright_{\mathcal{S}} \Gamma \vdash t : \sigma$ for some $\mathcal{S}$-testable typing $(\Gamma; \sigma)$.*

A type σ is **inhabited** in system $\mathcal{B}$, noted $\mathtt{inh}_{\mathcal{B}}(\sigma)$, if $\Pi \triangleright_{\mathcal{B}} \emptyset \vdash t : \sigma$ for some Π and t. For instance, in system $\mathcal{B}$, the type $[\,]$ is inhabited by any bang, use rule (bg) with no premises; the environment $\emptyset$ is trivially inhabited; the type $[\alpha] \Rightarrow [\alpha]$ is inhabited, see Figure 3. The term $\lambda x.!x$ is $\mathcal{B}$-testable because $\triangleright_{\mathcal{B}} \emptyset \vdash \lambda x.!x : [\,] \Rightarrow [\,]$ and $(\emptyset, [\,] \Rightarrow [\,])$ is $\mathcal{B}$-testable.

▶ **Lemma 6.** *Let $t \in \Lambda_!$ and $\mathtt{T}$ be a testing context. If $\triangleright_{\mathcal{B}} \emptyset \vdash \mathtt{T}\langle t\rangle : [\,]$, then $\triangleright_{\mathcal{B}} \Gamma \vdash t : \sigma$ with $\mathtt{inh}_{\mathcal{B}}(\Gamma)$ and $\mathtt{inh}_{\mathcal{B}}(\mathtt{args}_{\mathcal{B}}(\sigma))$.*

Inhabitation serves as a crucial tool to produce an observable from a typable term. As said before, any multitype assigned to a variable x by the environment Γ in the derivation of a meaningful term t should be inhabited. Hence, the environment Γ has to be inhabited. However, relying solely on the inhabitation of Γ is not sufficient, as illustrated by the typable term $\triangleright_{\mathcal{B}} \emptyset \vdash \lambda x.xx : [[\mathcal{M}] \Rightarrow \tau, \mathcal{M}] \Rightarrow \tau$, which, despite having a trivially inhabited environment, is dBang-meaningless. We thus also test the inhabitation of type arguments of the type σ of t. This therefore means that $\mathcal{B}$-testability is sufficient to ensure dBang-meaningfulness. Surprisingly, this actually provides a characterization of dBang-meaningfulness.

▶ **Theorem 7** (Logical Characterization). *Let $t \in \Lambda_!$: t is dBang-meaningful iff t is $\mathcal{B}$-testable.*

Now that we have a logical characterization of dBang-meaningfulness, we can reason about the consequences of equating all dBang-meaningless terms in a $\lambda_{\mathtt{dBang}}$-theory, that is, in a quotient of $\Lambda_!$ that roughly equates all terms with the same semantics. Formally, a $\lambda_{\mathtt{dBang}}$-**theory** is an equivalence $\equiv$ on $\Lambda_!$ containing $\to_{\mathrm{F}}$ and closed under full contexts. Let $\mathcal{H}_{\mathtt{dBang}}$ (also noted $\equiv_{\mathcal{H}_{\mathtt{dBang}}}$) be the smallest $\lambda_{\mathtt{dBang}}$-theory equating all dBang-meaningless terms. Theorem 7 entails that $\mathcal{H}_{\mathtt{dBang}}$ is **consistent**, that is, it does not equate all terms.

▶ **Proposition 8** (Consistency of $\mathcal{H}_{\mathtt{dBang}}$). *There exist $t, u \in \Lambda_!$ such that $t \not\equiv_{\mathcal{H}_{\mathtt{dBang}}} u$.*

Replacing dBang-meaningfulness by dBang-solvability would result in the loss of consistency. Indeed, take an arbitrary term $t \in \Lambda_!$ and the two dBang-unsolvable terms $!\Omega_!$ and $\Omega_!$ that the resulting (alternative) theory, written $\mathcal{H}^{\mathtt{solv}}_{\mathtt{dBang}}$, would equate. By contextuality, we would have $(\lambda x.t)\,!\Omega_! \equiv_{\mathcal{H}^{\mathtt{solv}}_{\mathtt{dBang}}} (\lambda x.t)\,\Omega_!$, and by reduction $t \equiv_{\mathcal{H}^{\mathtt{solv}}_{\mathtt{dBang}}} (\lambda x.t)\,!\Omega_!$ (suppose

$x \notin \mathtt{fv}(t)$) . Notice that $(\lambda x.t)\,\Omega_!$ is also dBang-unsolvable since the term $\Omega_!$ cannot be erased, thus $(\lambda x.t)\,\Omega_! \equiv_{\mathcal{H}^{\mathtt{solv}}_{\mathtt{dBang}}} \Omega_!$. By transitivity $t \equiv_{\mathcal{H}^{\mathtt{solv}}_{\mathtt{dBang}}} (\lambda x.t)\,!\Omega_! \equiv_{\mathcal{H}^{\mathtt{solv}}_{\mathtt{dBang}}} (\lambda x.t)\,\Omega_! \equiv_{\mathcal{H}^{\mathtt{solv}}_{\mathtt{dBang}}} \Omega_!$. Since t is arbitrary, we easily conclude that all terms are equated in $\mathcal{H}^{\mathtt{solv}}_{\mathtt{dBang}}$, making it inconsistent.

We also corroborate our definition of meaningfulness by proving that it fulfills a pair of genericity properties, and show that $\mathcal{H}_{\mathtt{dBang}}$ admits a unique maximal consistent extension $\mathcal{H}^{*}_{\mathtt{dBang}}$ (Section 4). Finally, we also show that dBang-meaningfulness, $\mathcal{H}_{\mathtt{dBang}}$ and $\mathcal{H}^{*}_{\mathtt{dBang}}$ subsume the well-established corresponding notions for dCBN and dCBV (Section 5).

4 Typed and Surface Genericity in dBang

In Section 3, we proved that dBang-meaningfulness is captured by typability in system $\mathcal{B}$ with some $\mathcal{B}$-testable typing. While this concise characterization formulated as "meaningfulness = typability + inhabitation" [26] provides a high level understanding, its practical manipulation might pose some challenges. Suppose we study some properties of a dBang-meaningful term t through the logical characterization (Theorem 7), thus having a type derivation $\Pi \rhd_{\mathcal{B}} \Gamma \vdash t : \sigma$ with $(\Gamma;\sigma)$ $\mathcal{B}$-testable. If we proceed by induction on Π, then there is no guarantee that all the judgments appearing in Π have $\mathcal{B}$-testable typings as well, which would make the reasoning awkward and the logical characterization of Theorem 7 difficult to exploit. But this is not the case. Upcoming Lemma 9 states that $\mathcal{B}$-testability propagates bottom-up: if the conclusion of a derivation Π has a $\mathcal{B}$-testable typing, then so does every other judgment in Π.

We write $\Pi \rhd_{\mathcal{B}_{\mathtt{m}}} \Gamma \vdash t : \sigma$ if $\Pi \rhd_{\mathcal{B}} \Gamma \vdash t : \sigma$ and each judgment in Π is $\mathcal{B}$-testable, and $\Pi \rhd_{\mathcal{B}_{\mathtt{m}}} t$ if $\Pi \rhd_{\mathcal{B}_{\mathtt{m}}} \Gamma \vdash t : \sigma$ holds for some typing $(\Gamma;\sigma)$.

▶ **Lemma 9.** *Let $t \in \Lambda_!$. Then $\Pi \rhd_{\mathcal{B}} \Gamma \vdash t : \sigma$ with $(\Gamma;\sigma)$ $\mathcal{B}$-testable iff $\Pi \rhd_{\mathcal{B}_{\mathtt{m}}} \Gamma \vdash t : \sigma$.*

Proof. ($\Leftarrow$): Trivial. ($\Rightarrow$): By an induction on Π. ◀

We can therefore easily use the logical characterization of dBang-meaningfulness to prove the following first *genericity* result for dBang: in a dBang-meaningful term s, a dBang-meaningless subterm can be replaced by any term, without impacting the typing of s.

▶ **Theorem 10** (Typed Genericity)**.** *Let $t \in \Lambda_!$ be dBang-meaningless and $\mathtt{F}$ be a full context. If $\rhd_{\mathcal{B}_{\mathtt{m}}} \Gamma \vdash \mathtt{F}\langle t\rangle : \sigma$, then $\rhd_{\mathcal{B}_{\mathtt{m}}} \Gamma \vdash \mathtt{F}\langle u\rangle : \sigma$ for all $u \in \Lambda_!$.*

Proof. By induction on $\mathtt{F}$, using both Theorem 7 and Lemma 9. ◀

This proof relies on the fact that the dBang-meaningless subterm t cannot be explicitly typed in any of the judgments of Π, as typing t in $\mathcal{B}_{\mathtt{m}}$ is equivalent to being dBang-meaningful (by Theorem 7 and Lemma 9). Thus, typed genericity fails when weakening the hypothesis from $\mathcal{B}_{\mathtt{m}}$-typability to $\mathcal{B}$-typability. For example, given the dBang-meaningless term $t = xx$ and the context $\mathtt{F} = y\,\diamond$, $\mathtt{F}\langle t\rangle$ is $\mathcal{B}$-typable as witnessed by $\rhd_{\mathcal{B}} y : [\mathcal{N} \Rightarrow \alpha]\,, x : [\mathcal{M} \Rightarrow \mathcal{N}, \mathcal{M}] \vdash \mathtt{F}\langle t\rangle : \alpha$ – note that the type of x is not inhabited – while $\mathtt{F}\langle \Omega_!\rangle = y\,\Omega_!$ is not $\mathcal{B}$-typable.

As a consequence of typed genericity, we can now prove a qualitative surface genericity result, stating that dBang-meaningless subterms have no bearing on the significance of dBang-meaningful terms: in a dBang-meaningful term s, a dBang-meaningless subterm can be replaced by any term, still keeping s dBang-meaningful. We call this genericity result *surface*, despite it universally quantifies over full contexts, as dBang-meaningful is defined in terms of surface reduction. The corresponding results for dCBN and dCBV are also called *surface* in [15] and *light* in [10], they are both later generalized to a *stratified* notion in [15].

▶ **Corollary 11** (Qualitative Surface Genericity)**.** *Let $\mathtt{F}$ be a full context. If $\mathtt{F}\langle t\rangle$ is dBang-meaningful for some dBang-meaningless $t \in \Lambda_!$, then $\mathtt{F}\langle u\rangle$ is dBang-meaningful for all $u \in \Lambda_!$.*

Proof. Let $u \in \Lambda_!$. As $\mathtt{F}\langle t\rangle$ is dBang-meaningful, then $\Pi \rhd_{\mathcal{B}_m} \mathtt{F}\langle t\rangle$ holds for some Π by Theorem 7 and Lemma 9. As t is dBang-meaningless, then $\Pi' \rhd_{\mathcal{B}_m} \mathtt{F}\langle u\rangle$ holds for some Π' by Theorem 10, and hence $\mathtt{F}\langle u\rangle$ is dBang-meaningful by Theorem 7 and Lemma 9. ◀

As for consistency, surface genericity fails when replacing dBang-meaningfulness with dBang-solvability. Indeed, consider the full context $\mathtt{F} := (\lambda y.x)\,\diamond$ and the two dBang-unsolvable terms $t = !\Omega_!$ and $u = \Omega_!$. One then has that $\mathtt{F}\langle t\rangle = (\lambda y.x)\,!\Omega_! \to_{\mathrm{S}}^* x$ is trivially dBang-solvable, while $\mathtt{F}\langle u\rangle = (\lambda y.x)\,\Omega_!$ is not, as the term $\Omega_!$ cannot be erased.

Genericity is a sanity check on meaningfulness: it holds only if all dBang-meaningless terms are *truly* meaningless. Still, some truly meaningless terms might be misinterpreted as dBang-meaningful. Indeed, when crafting a notion of dBang-meaningless that would satisfy genericity, one might not take *all* truly meaningless terms. The $\lambda_{\mathtt{dBang}}$-theory $\mathcal{H}^*_{\mathtt{dBang}}$ is introduced to avoid that. Let $\mathcal{H}^*_{\mathtt{dBang}}$, also noted $\equiv_{\mathcal{H}^*_{\mathtt{dBang}}}$, be the relation on $\Lambda_!$ defined by:

$$\mathcal{H}^*_{\mathtt{dBang}} := \{(t,u) \mid \forall\ \mathtt{F} \text{ full context, } \mathtt{F}\langle t\rangle \text{ dBang-meaningful} \Leftrightarrow \mathtt{F}\langle u\rangle \text{ dBang-meaningful}\}$$

The theory $\mathcal{H}^*_{\mathtt{dBang}}$ equates *more* than $\mathcal{H}_{\mathtt{dBang}}$. For example, let $t = x[x\backslash z][y\backslash z]$ and $u = x[y\backslash z][x\backslash z]$: it can be shown that $t \not\equiv_{\mathcal{H}_{\mathtt{dBang}}} u$ while $t \equiv_{\mathcal{H}^*_{\mathtt{dBang}}} u$ due to Theorem 7 since t and u (and so $\mathtt{F}\langle t\rangle$ and $\mathtt{F}\langle u\rangle$ for any full context $\mathtt{F}$) are $\mathcal{B}$-typable by exactly the same typings.

▶ Remark 12. In $\mathcal{H}^*_{\mathtt{dBang}}$, a term reducing to a bang will only be equated to terms which also reduce to bangs. This can be formally proved using a property stating that neutral normal forms can create clashes via a single substitution, technical details can be found in [50].

We expect $\mathcal{H}^*_{\mathtt{dBang}}$ to extend the theory $\mathcal{H}_{\mathtt{dBang}}$. Moreover, to check that all truly meaningless terms are actually dBang-meaningless, we also want this theory to be **maximal**, meaning that no more terms can additionally be equated without compromising consistency.

▶ **Theorem 13.** *$\mathcal{H}^*_{dBang}$ is the unique maximal consistent λ_{dBang}-theory containing $\mathcal{H}_{dBang}$.*

We now show that the theory $\mathcal{H}^*_{\mathtt{dBang}}$ coincides with the well-known notion of observational equivalence in the literature. Observational equivalence roughly equates terms having the same operational behavior (*i.e.* reduction to an observable) in any context. The fact that $\mathcal{H}^*_{\mathtt{dBang}}$ and observational equivalence coincide means that two different approaches to define a semantics in dBang actually coincide. This further backs up the idea that what we call dBang-meaningfulness appropriately represents meaningfulness in dBang.

▶ **Definition 14** (Observational Equivalence). *Let $t, u \in \Lambda_!$, then t and u are **open-observational equivalent** (resp. **observational equivalent**), noted $t \cong^o u$ (resp. $t \cong u$) if for every full context $\mathtt{F}$ (resp. full context $\mathtt{F}$ such that $\mathtt{F}\langle t\rangle$ and $\mathtt{F}\langle u\rangle$ are closed), $\mathtt{F}\langle t\rangle \to_{\mathrm{S}}^* !t'$ for some $t' \in \Lambda_!$ iff $\mathtt{F}\langle u\rangle \to_{\mathrm{S}}^* !u'$ for some $u' \in \Lambda_!$.*

Note that, differently from $\cong$, $\cong^o$ quantifies over all full contexts and not only on closing full contexts, hence $\cong^o \subseteq \cong$. Finally, we now prove that the $\lambda_{\mathtt{dBang}}$-theory $\mathcal{H}^*_{\mathtt{dBang}}$ actually coincides with the observational equivalences $\cong$ and $\cong^o$.

▶ **Theorem 15.** *Let $t, u \in \Lambda_!$, then (1) $t \cong u$ iff (2) $t \cong^o u$ iff (3) $t \equiv_{\mathcal{H}^*_{dBang}} u$.*

Proof. Let $t, u \in \Lambda_!$. Let us show that (3) ⇒ (2) ⇒ (1) ⇒ (3).

- (3) ⇒ (2): Let $t \equiv_{\mathcal{H}^*_{\mathtt{dBang}}} u$. Suppose $\mathtt{F}$ is an arbitrary full context such that $\mathtt{F}\langle t\rangle \to_{\mathrm{S}}^* !t'$ for some $t' \in \Lambda_!$. Since $\mathcal{H}^*_{\mathtt{dBang}}$ is a $\lambda_{\mathtt{dBang}}$-theory (Theorem 13) then it is contextual and hence $\mathtt{F}\langle t\rangle \equiv_{\mathcal{H}^*_{\mathtt{dBang}}} \mathtt{F}\langle u\rangle$. By Remark 12, $\mathtt{F}\langle u\rangle \to_{\mathrm{S}}^* !u'$ for some $u' \in \Lambda_!$. Therefore, $t \cong u$.
- (2) ⇒ (1): Immediate.
- (1) ⇒ (3): We can easily prove that $\cong$ is a consistent $\lambda_{\mathtt{dBang}}$-theory. As (3) ⇒ (2) ⇒ (1), we have $\cong \supseteq \mathcal{H}^*_{\mathtt{dBang}} \supseteq \mathcal{H}_{\mathtt{dBang}}$ (the last inclusion holds by Theorem 13). By maximality of $\mathcal{H}^*_{\mathtt{dBang}}$ (Theorem 13), then necessarily $\cong \subseteq \mathcal{H}^*_{\mathtt{dBang}}$. ◀

5 Subsuming CBN and CBV Meaningfulness

In this section we show that the notions of meaningfulness for dCBN and dCBV in the literature [15] are subsumed by the one proposed in Section 3 for dBang. We also deduce surface genericity for dCBN and dCBV as a consequence of surface genericity for dBang.

5.1 dCBN and dCBV Calculi

Both dCBN [7, 8, 1] and dCBV [11] are specified using ES and action at a distance, as explained in Section 2.1 for dBang. Both dCBN and dCBV share the same term syntax. The sets Λ of **terms** and Υ of **values** are inductively defined below.

$$\textbf{(Terms)}\quad t,u ::= v \mid t\,u \mid t[x\backslash u] \qquad\qquad \textbf{(Values)}\quad v ::= x \mid \lambda x.t$$

From now on, we set $\mathtt{I} := \lambda z.z$, $\Delta := \lambda x.xx$, and $\Omega := \Delta\Delta$. Note that the syntax contains neither der nor !. The distinction between terms and values is irrelevant in dCBN but crucial in dCBV. The two calculi also share the same **list contexts** $\mathtt{L_N}, \mathtt{L_V}$ and **full contexts** $\mathtt{F_N}, \mathtt{F_V}$, but use specialized **surface contexts** $\mathtt{S_N}$ and $\mathtt{S_V}$ for dCBN and dCBV, respectively. Again, contexts can be seen as terms with exactly one **hole** $\diamond$ and are inductively defined below.

$$\begin{array}{rrcl}
\textbf{(List Contexts)} & \mathtt{L_N}, \mathtt{L_V} & ::= & \diamond \mid \mathtt{L_N}[x\backslash t] \\
\textbf{(dCBN Surface Contexts)} & \mathtt{S_N} & ::= & \diamond \mid \mathtt{S_N}\,t \mid \lambda x.\mathtt{S_N} \mid \mathtt{S_N}[x\backslash t] \\
\textbf{(dCBV Surface Contexts)} & \mathtt{S_V} & ::= & \diamond \mid \mathtt{S_V}\,t \mid t\,\mathtt{S_V} \mid \mathtt{S_V}[x\backslash t] \mid t[x\backslash \mathtt{S_V}] \\
\textbf{(Full Contexts)} & \mathtt{F_N}, \mathtt{F_V} & ::= & \diamond \mid \mathtt{F_N}\,t \mid t\,\mathtt{F_N} \mid \lambda x.\mathtt{F_N} \mid \mathtt{F_N}[x\backslash t] \mid t[x\backslash \mathtt{F_N}]
\end{array}$$

We now consider the following *rewrite rules*:

$$\mathtt{L_N}\langle\lambda x.t\rangle\, u \mapsto_{\mathtt{dB}} \mathtt{L_N}\langle t[x\backslash u]\rangle \qquad t[x\backslash u] \mapsto_{\mathtt{s}} t\{x\backslash u\} \qquad t[x\backslash \mathtt{L_V}\langle v\rangle] \mapsto_{\mathtt{sV}} \mathtt{L_V}\langle t\{x\backslash v\}\rangle$$

Rules dB and sV are both capture-free: no free variable of u (resp. t) is captured by the list context $\mathtt{L_N}$ (resp. $\mathtt{L_V}$). The differences between dCBN and dCBV are in the previous notions of *surface* contexts, and in the rewrite rules. The **dCBN surface reduction** $\to_{\mathtt{S_N}}$ is the union of the dCBN surface closure of rewrite rules dB and s, while the **dCBV surface reduction** $\to_{\mathtt{S_V}}$ is the union of the dCBV surface closure of the rewrite rules dB and sV. Finally, we use $\to^*_{\mathtt{S_N}}$ (resp. $\to^*_{\mathtt{S_V}}$) to denote the reflexive-transitive closure of the relation $\to_{\mathtt{S_N}}$ (resp. $\to_{\mathtt{S_V}}$).

▶ **Example 16.** For example, $t_0 := (\lambda x.yxx)(\mathtt{II}) \to_{\mathtt{S_N}} (yxx)[x\backslash \mathtt{II}] \to_{\mathtt{S_N}} y(\mathtt{II})(\mathtt{II}) =: t_1$ and $t_0 = (\lambda x.yxx)(\mathtt{II}) \to_{\mathtt{S_V}} (yxx)[x\backslash \mathtt{II}] \to_{\mathtt{S_V}} (yxx)[x\backslash z[z\backslash \mathtt{I}]] \to_{\mathtt{S_V}} (yxx)[x\backslash \mathtt{I}] \to_{\mathtt{S_V}} y\mathtt{II} =: t_2$.

The dCBN surface reduction is (a non-deterministic diamond variant of) the well-known *head* reduction [20], and dCBV surface reduction is the *weak* reduction not reducing under λ's.

The quantitative type systems $\mathcal{N}$ for dCBN and $\mathcal{V}$ for dCBV are presented in Figures 4 and 5, respectively. **Types** and **judgments** are the same as for system $\mathcal{B}$. A derivation Π in system $\mathcal{N}$ with conclusion $\Gamma \vdash t : \sigma$ is noted $\Pi \rhd_{\mathcal{N}} \Gamma \vdash t : \sigma$; we write $\rhd_{\mathcal{N}} \Gamma \vdash t : \sigma$ if there is a derivation $\Pi \rhd_{\mathcal{N}} \Gamma \vdash t : \sigma$. We use similar notations for system $\mathcal{V}$.

The salient property of type systems $\mathcal{N}$ and $\mathcal{V}$ is characterizing normalization in dCBN and dCBV, respectively.

▶ **Lemma 17** ([23, 24]). *Let $t \in \Lambda$, then:*

- *t is dCBN surface normalizing iff it is $\mathcal{N}$-typable.*
- *t is dCBV surface normalizing iff it is $\mathcal{V}$-typable.*

$$\frac{}{x : [\sigma] \vdash x : \sigma}\ (var) \qquad \frac{\Gamma \vdash t : [\tau_i]_{i\in I} \Rightarrow \sigma \quad (\Delta_i \vdash u : \tau_i)_{i\in I} \quad I \text{ finite}}{\Gamma +_{i\in I} \Delta_i \vdash t\, u : \sigma}\ (app)$$

$$\frac{\Gamma, x : \mathcal{M} \vdash t : \sigma}{\Gamma \vdash \lambda x.t : \mathcal{M} \Rightarrow \sigma}\ (abs) \qquad \frac{\Gamma, x : [\tau_i]_{i\in I} \vdash t : \sigma \quad (\Delta_i \vdash u : \tau_i)_{i\in I} \quad I \text{ finite}}{\Gamma +_{i\in I} \Delta_i \vdash t[x\backslash u] : \sigma}\ (es)$$

Figure 4 Type System $\mathcal{N}$ for the dCBN-calculus.

$$\frac{}{x : \mathcal{M} \vdash x : \mathcal{M}}\ (var) \qquad \frac{\Gamma \vdash t : [\mathcal{M} \Rightarrow \sigma] \quad \Delta \vdash u : \mathcal{M}}{\Gamma + \Delta \vdash t\, u : \sigma}\ (app)$$

$$\frac{(\Gamma_i, x : \mathcal{M}_i \vdash t : \sigma_i)_{i\in I} \quad I \text{ finite}}{+_{i\in I}\Gamma_i \vdash \lambda x.t : [\mathcal{M}_i \Rightarrow \sigma_i]_{i\in I}}\ (abs) \qquad \frac{\Gamma, x : \mathcal{M} \vdash t : \sigma \quad \Delta \vdash u : \mathcal{M}}{\Gamma + \Delta \vdash t[x\backslash u] : \sigma}\ (es)$$

Figure 5 Type System $\mathcal{V}$ for the dCBV-calculus.

Both dCBN and dCBV can be embedded into dBang by decorating each term with the ! and der modalities. The embedding $\cdot^{\mathtt{n}}$ for dCBN is standard, while various embeddings $\cdot^{\mathtt{v}}$ for dCBV have been proposed in the literature [44, 57, 58, 46, 23, 24, 14], each with its own strengths and weaknesses. In this work, we use the embeddings from [23, 24] defined below:

$$\begin{array}{rcl@{\qquad}rcl} x^{\mathtt{n}} & := & x & x^{\mathtt{v}} & := & !x \\ (\lambda x.t)^{\mathtt{n}} & := & \lambda x.t^{\mathtt{n}} & (\lambda x.t)^{\mathtt{v}} & := & !\lambda x.t^{\mathtt{v}} \\ (tu)^{\mathtt{n}} & := & t^{\mathtt{n}}\, !u^{\mathtt{n}} & (tu)^{\mathtt{v}} & := & \begin{cases} \mathtt{L}\langle s\rangle\, u^{\mathtt{v}} & \text{if } t^{\mathtt{v}} = \mathtt{L}\langle !s\rangle \\ \mathtt{der}(t^{\mathtt{v}})\, u^{\mathtt{v}} & \text{otherwise} \end{cases} \\ (t[x\backslash u])^{\mathtt{n}} & := & t^{\mathtt{n}}[x\backslash !u^{\mathtt{n}}] & (t[x\backslash u])^{\mathtt{v}} & := & t^{\mathtt{v}}[x\backslash u^{\mathtt{v}}] \end{array}$$

These translations are extended to contexts as expected by setting $\diamond^{\mathtt{n}} := \diamond$ and $\diamond^{\mathtt{v}} := \diamond$.

▶ **Example 18.** Recalling Example 16, one has $t_0^{\mathtt{n}} = (\lambda x.y\, !x\, !x)\, !(\mathtt{I}_!!\mathtt{I}_!)$, $t_1^{\mathtt{n}} = y\, !(\mathtt{I}_!!\mathtt{I}_!)\, !(\mathtt{I}_!!\mathtt{I}_!)$, $t_0^{\mathtt{v}} = (\lambda x.(\mathtt{der}(y\, !x)\, !x))(\mathtt{I}_!\, !\mathtt{I}_!)$ and $t_2^{\mathtt{v}} = \mathtt{der}(y\, !\mathtt{I}_!)\, !\mathtt{I}_!$.

Let us give some intuition on these embeddings. In dCBN, any argument (right-hand side of application or substitution) can be erased/duplicated, just as bang terms in the dBang-calculus, so that arguments must be translated to bang terms. In dCBV, only values can be erased/duplicated so that values – and only values – must be translated to bang terms. However, this remark alone is not sufficient to achieve a dCBV embedding enjoying good properties, and in particular to translate dCBV-normal forms to dBang-normal forms. The translation of applications is precisely designed in order to guarantee this property.

These embeddings preserve reductions, which will allows us to show that meaningfulness if preserved through embedding (Theorems 25 and 30).

▶ **Lemma 19** (Simulation [23, 24]). *Let $t, u \in \Lambda$.*

1. *If $t \to_{\mathtt{S}_{\mathtt{N}}}^* u$ then $t^{\mathtt{n}} \to_{\mathtt{S}}^* u^{\mathtt{n}}$.*
2. *If $t \to_{\mathtt{S}_{\mathtt{V}}}^* u$ then $t^{\mathtt{v}} \to_{\mathtt{S}}^* u^{\mathtt{v}}$.*

▶ **Example 20.** In Example 16, we showed that $t_0 \to_{\mathtt{S}_{\mathtt{N}}}^* t_1$ and $t_0 \to_{\mathtt{S}_{\mathtt{V}}}^* t_2$. Recalling Example 18, one has $t_0^{\mathtt{n}} \to_{\mathtt{S}} (y\, !x\, !x)[x\backslash !(\mathtt{I}_!!\mathtt{I}_!)] \to_{\mathtt{S}} t_1^{\mathtt{n}}$ and $t_0^{\mathtt{v}} \to_{\mathtt{S}} (\mathtt{der}(y\, !x)\, !x)[x\backslash \mathtt{I}_!\, !\mathtt{I}_!] \to_{\mathtt{S}} (\mathtt{der}(y\, !x)\, !x)[x\backslash (!z)[z\backslash !\mathtt{I}_!]] \to_{\mathtt{S}} (\mathtt{der}(y\, !x)\, !x)[x\backslash !\mathtt{I}_!] \to_{\mathtt{S}} t_2^{\mathtt{v}}$.

As the dCBV-embedding uses der, some d!-step might be needed in the simulation process.

These embeddings also preserve typing, which will make possible to project dBang meaningfulness and surface genericity onto dCBN and dCBV. More precisely, the two embeddings are proven to be sound and complete with respect to system $\mathcal{B}$.

▶ **Proposition 21** ([23, 24]). *Let $t \in \Lambda$ and $(\Gamma; \sigma)$ be a typing.*
1. *One has $\triangleright_{\mathcal{N}} \Gamma \vdash t : \sigma$ if and only if $\triangleright_{\mathcal{B}} \Gamma \vdash t^{\mathtt{n}} : \sigma$.*
2. *One has $\triangleright_{\mathcal{V}} \Gamma \vdash t : \sigma$ if and only if $\triangleright_{\mathcal{B}} \Gamma \vdash t^{\mathtt{v}} : \sigma$.*

A straightforward corollary is that dCBN and dCBV inhabitation properties are well subsumed in dBang, as illustrated in [13]. In simpler words, any type inhabited in dCBN (resp. dCBV) is also inhabited in dBang. As expected, the converse is false.

In dCBV and dBang, typing an arbitrary term and typing an argument is similar, as it can be seen in the right premise $\Delta \vdash u : \mathcal{M}$ of the typing rules (app) and (es) of systems $\mathcal{V}$ and $\mathcal{B}$. This is not the case in dCBN, as the right premise of the (app) and (es) rules of system $\mathcal{N}$ requires, not a *single* derivation, but a finite *set* $(\Delta_i \vdash u : \tau_i)_{i \in I}$ of typing derivation for the same term u. In the logical characterization (Theorem 7), we check that arguments of a given type can be inhabited. We therefore need to reflect the typability of arguments – rather than typability of arbitrary terms – in the definition of dCBN inhabitation.

▶ **Definition 22.** *In system $\mathcal{N}$, a non-multitype σ is **inhabited**, noted $\mathtt{inh}_{\mathcal{N}}(\sigma)$, if $\Pi \triangleright_{\mathcal{N}} \emptyset \vdash t : \sigma$ for some Π and t. A multitype $[\tau_i]_{i \in I}$ is **inhabited** in system $\mathcal{N}$, noted $\mathtt{inh}_{\mathcal{N}}([\tau_i]_{i \in I})$ if there exists $u \in \Lambda$ such that for each $i \in I$, $\triangleright_{\mathcal{N}} \emptyset \vdash u : \tau_i$.*

*In system $\mathcal{V}$, a type σ is **inhabited**, noted $\mathtt{inh}_{\mathcal{V}}(\sigma)$, if $\Pi \triangleright_{\mathcal{V}} \emptyset \vdash t : \sigma$ for some Π and t.*

In particular, the type $[\,]$ is inhabited in both dCBN and dCBV (*i.e.* $\mathtt{inh}_{\mathcal{N}}([\,])$ and $\mathtt{inh}_{\mathcal{V}}([\,])$). Similarly, the environment $\emptyset$ is also trivially inhabited in both (*i.e.* $\mathtt{inh}_{\mathcal{N}}(\emptyset)$ and $\mathtt{inh}_{\mathcal{V}}(\emptyset)$).

5.2 dCBN Meaningfulness and Surface Genericity

In this subsection, our attention shifts towards the dCBN-calculus, where we show that its notion of meaningfulness is subsumed by that of dBang. This observation enables us to project the surface genericity theorem accordingly. We start by introducing dCBN-meaningfulness.

▶ **Definition 23.** *A term $t \in \Lambda$ is **dCBN-meaningful** if there is a testing context $\mathtt{T_N}$ such that $\mathtt{T_N}\langle t\rangle \to^*_{\mathtt{S_N}} \mathtt{I}$, where testing contexts are defined by $\mathtt{T_N} ::= \diamond \mid \mathtt{T_N}\, u \mid (\lambda x.\mathtt{T_N})\, u$.*[2]

For example $t = x(\lambda y.\Omega)$ is dCBN-meaningful as $\mathtt{T_N}\langle t\rangle \to^*_{\mathtt{S_N}} \mathtt{I}$ for $\mathtt{T_N} = (\lambda x.\diamond)(\lambda z.\mathtt{I})$, while Ω and $\lambda x.\Omega$ are dCBN-meaningless as for whatever testing context Ω and $\lambda x.\Omega$ are plugged into, Ω will not be erased. According to the definition of dCBN-meaningfulness, it is natural to define the types of observable terms in dCBN as the identity types, *i.e.* $\mathcal{T}^{\mathtt{obs}}_{\mathcal{N}} := \{[\sigma] \Rightarrow \sigma \mid \sigma \text{ type}\}$.

Unlike dBang, dCBN-meaningfulness can be characterized both *operationally*, through surface normalizability, and *logically*, through typability in system $\mathcal{N}$. Moreover, this logical characterization turns out to be equivalent to *$\mathcal{N}$-testability*, meaning that dCBN-meaningfulness can also be characterized via typability and inhabitation, as already observed in [27].

[2] Usually, dCBN-meaningfulness (aka *solvability*) is defined using contexts of the form $(\lambda x_1 \dots x_m.\diamond)N_1 \dots N_n$ $(m, n \geq 0)$ [19, 20, 71], instead of testing contexts. It is easy to check that the two definitions are equivalent in dCBN. The benefit of our definition is that the same testing contexts are also used to define dCBV-meaningfulness (Section 5.3).

▶ **Theorem 24** (Characterizations of dCBN-Meaningfulness [29, 27, 23]). *Let $t \in \Lambda$.*
1. ***(Operational)*** *t is dCBN-meaningful iff t is dCBN surface-normalizing.*
2. ***(Logical)*** *(1) t is dCBN-meaningful iff (2) t is $\mathcal{N}$-typable iff (3) t is $\mathcal{N}$-testable.*

Thanks to the specific shape of dCBN-normal forms, we can always type a dCBN-meaningful term t by a typing $(\Gamma;\sigma)$ such that the *non-empty* multitypes in Γ and $\mathtt{args}_{\mathcal{N}}(\sigma)$ are of the form $[[\,] \Rightarrow \cdots [\,] \Rightarrow [\alpha] \Rightarrow \alpha]$. These types are trivially inhabited by erasers of the form $\lambda x_1.\cdots\lambda x_n.\mathtt{I}$, used to prove that $\mathcal{N}$-typability implies dCBN-meaningfulness.

Having an operational characterization of meaningfulness seems to point out that transforming a result into something observable is a trivial operation in dCBN. Indeed, using simulation (Lemma 19.2), we easily show that dCBN-meaningful is preserved by the dCBN-embedding, thus confirming this intuition. Moreover, and thanks to the logical characterization (Theorem 24.2), we show that the converse also holds, yielding the following result.

▶ **Theorem 25.** *Let $t \in \Lambda$, then t is dCBN-meaningful iff $t^{\mathtt{n}}$ is dBang-meaningful.*

Proof.
($\Rightarrow$) We present here an operational proof. Let t be dCBN-meaningful, thus $\mathtt{T}_{\mathtt{N}}\langle t\rangle \to_{\mathtt{S}_{\mathtt{N}}}^{*} \mathtt{I}$ for some testing context $\mathtt{T}_{\mathtt{N}}$. By induction on $\mathtt{T}_{\mathtt{N}}$, one has that $(\mathtt{T}_{\mathtt{N}}\langle t\rangle)^{\mathtt{n}} = \mathtt{T}_{\mathtt{N}}^{\mathtt{n}}\langle t^{\mathtt{n}}\rangle$. By simulation (Lemma 19.1), one deduces that $\mathtt{T}_{\mathtt{N}}^{\mathtt{n}}\langle t^{\mathtt{n}}\rangle \to_{\mathtt{S}}^{*} \lambda x.x$ thus $\mathtt{T}_{\mathtt{N}}^{\mathtt{n}}\langle t^{\mathtt{n}}\rangle\,!!y \to_{\mathtt{S}}^{*} (\lambda x.x)!!y \to_{\mathtt{S}}^{*} !y$. Notice that $\mathtt{T}_{\mathtt{N}}^{\mathtt{n}}!!y$ is a dBang-testing context. We thus conclude that $t^{\mathtt{n}}$ is dBang-meaningful.
($\Leftarrow$) Let $t^{\mathtt{n}}$ be dBang-meaningful, then using Theorem 7, it is $\mathcal{B}$-testable and thus $\mathcal{B}$-typable. By Proposition 21.1, t is $\mathcal{N}$-typable and hence t is dCBN-meaningful by Theorem 24. ◀

Observe for example that $\mathtt{I}$ and $\mathtt{I}^{\mathtt{n}} = \mathtt{I}_!$ are both dCBN/dBang-meaningful while Ω and $\Omega^{\mathtt{n}} = \Omega_!$ are both dCBN/dBang-meaningless.

Theorem 25 states that dCBN-meaningfulness precisely aligns with dBang-meaningfulness on its image via $\cdot^{\mathtt{n}}$, strengthening the idea that these two notions are adequately chosen. Thanks to Theorem 25, we can now project surface genericity from dBang to dCBN.

▶ **Theorem 26** (dCBN Qualitative Surface Genericity). *Let $\mathtt{F}_{\mathtt{N}}$ be a full context. If $\mathtt{F}_{\mathtt{N}}\langle t\rangle$ is dCBN-meaningful for some dCBN-meaningless $t \in \Lambda$, then $\mathtt{F}_{\mathtt{N}}\langle u\rangle$ is dCBN-meaningful for every $u \in \Lambda$.*

Proof. Let $t \in \Lambda$ be dCBN-meaningless and $\mathtt{F}_{\mathtt{N}}$ be a full context. Suppose that $\mathtt{F}_{\mathtt{N}}\langle t\rangle$ is dCBN-meaningful: by Theorem 25 and since $(\mathtt{F}_{\mathtt{N}}\langle t\rangle)^{\mathtt{n}} = \mathtt{F}_{\mathtt{N}}^{\mathtt{n}}\langle t^{\mathtt{n}}\rangle$ (simple induction on $\mathtt{F}_{\mathtt{N}}$), $\mathtt{F}_{\mathtt{N}}^{\mathtt{n}}\langle t^{\mathtt{n}}\rangle$ is dBang-meaningful, and $t^{\mathtt{n}}$ is dBang-meaningless. By Corollary 11, for any $u \in \Lambda$, $\mathtt{F}_{\mathtt{N}}^{\mathtt{n}}\langle u^{\mathtt{n}}\rangle = (\mathtt{F}_{\mathtt{N}}\langle u\rangle)^{\mathtt{n}}$ is dBang-meaningful, and hence $\mathtt{F}_{\mathtt{N}}\langle u\rangle$ is dCBN-meaningful using Theorem 25. ◀

We now discuss some crucial consequences of our previous results, captured by the use of $\lambda_{\mathtt{dCBN}}$-theories. A **$\lambda_{\mathtt{dCBN}}$-theory** is an equivalence $\equiv$ on Λ containing $\to_{\mathtt{F}_{\mathtt{N}}}$ and closed under full contexts. Let $\mathcal{H}_{\mathtt{dCBN}}$ (also noted $\equiv_{\mathcal{H}_{\mathtt{dCBN}}}$) be the smallest $\lambda_{\mathtt{dCBN}}$-theory equating all dCBN-meaningless terms, and let $\mathcal{H}^{*}_{\mathtt{dCBN}}$ be defined as follows:

$$\mathcal{H}^{*}_{\mathtt{dCBN}} := \{(t,u) \mid \forall\, \mathtt{F}_{\mathtt{N}} \text{ full context}, \mathtt{F}_{\mathtt{N}}\langle t\rangle \text{ dCBN-meaningful} \Leftrightarrow \mathtt{F}\langle u\rangle \text{ dBang-meaningful}\}$$

As for dBang, $\mathcal{H}^{*}_{\mathtt{dCBN}}$ is the maximal consistent $\lambda_{\mathtt{dCBN}}$-theory containing $\mathcal{H}_{\mathtt{dCBN}}$ and it coincides with observational equivalence in dCBN (see [15]). Thanks to the preservation of meaningfulness via the dCBN-embedding $\cdot^{\mathtt{n}}$ (Theorem 25), we can actually relate the theories $\mathcal{H}_{\mathtt{dBang}}$ and $\mathcal{H}^{*}_{\mathtt{dBang}}$ (in dBang) to the corresponding ones in dCBN, that is, $\mathcal{H}_{\mathtt{dCBN}}$ and $\mathcal{H}^{*}_{\mathtt{dCBN}}$ respectively.

▶ **Theorem 27.** *Let $t, u \in \Lambda$.*
1. *If $t \equiv_{\mathcal{H}_{\mathtt{dCBN}}} u$ then $t^{\mathtt{n}} \equiv_{\mathcal{H}_{\mathtt{dBang}}} u^{\mathtt{n}}$.*
2. *If $t^{\mathtt{n}} \equiv_{\mathcal{H}^{*}_{\mathtt{dBang}}} u^{\mathtt{n}}$ then $t \equiv_{\mathcal{H}^{*}_{\mathtt{dCBN}}} u$.*

Proof.

1. Immediate consequence of Theorem 25 and Lemma 19.1.
2. Let $t, u \in \Lambda$ such that $t^{\mathtt{n}} \equiv_{\mathcal{H}^*_{\mathtt{dBang}}} u^{\mathtt{n}}$. Let $\mathtt{F}_{\mathtt{N}}$ be a full context and suppose that $\mathtt{F}_{\mathtt{N}}\langle t\rangle$ is dCBN-meaningful. Using Theorem 25, one deduces that $(\mathtt{F}_{\mathtt{N}}\langle t\rangle)^{\mathtt{n}} = \mathtt{F}^{\mathtt{n}}_{\mathtt{N}}\langle t^{\mathtt{n}}\rangle$ is dBang-meaningful. Since $t^{\mathtt{n}} \equiv_{\mathcal{H}^*_{\mathtt{dBang}}} u^{\mathtt{n}}$, one has that $\mathtt{F}^{\mathtt{n}}_{\mathtt{N}}\langle u^{\mathtt{n}}\rangle = (\mathtt{F}_{\mathtt{N}}\langle u\rangle)^{\mathtt{n}}$ is dBang-meaningful. Using Theorem 25, one concludes that $\mathtt{F}_{\mathtt{N}}\langle u\rangle$ is dCBN-meaningful and therefore $t \equiv_{\mathcal{H}^*_{\mathtt{dCBN}}} u$. ◀

We strongly conjecture that the converse of Theorem 27.1 also holds. Perhaps unexpectedly, the converse of Theorem 27.2 is actually false. Indeed, η-expansion is included in $\mathcal{H}^*_{\mathtt{dCBN}}$ (see [20]) but not in $\mathcal{H}^*_{\mathtt{dBang}}$ thus $x \equiv_{\mathcal{H}^*_{\mathtt{dCBN}}} \lambda y.xy$ but $x^{\mathtt{n}} = x \not\equiv_{\mathcal{H}^*_{\mathtt{dCBN}}} \lambda y.x!y = (\lambda y.xy)^{\mathtt{n}}$: the context $\mathtt{F} = \diamond[x\backslash !!w]$ separates x and $\lambda y.x!y$. However, through Theorem 15, this phenomenon is not so surprising as it tells us that the dCBN observational equivalence does not coincide with dBang observational equivalence on the image of $\cdot^{\mathtt{n}}$, since dBang is a finer language than dCBN, with more contexts to separate terms operationally.

5.3 dCBV Meaningfulness and Surface Genericity

We now move to the dCBV-calculus, where we show that its notion of meaningfulness is subsumed by that of the dBang-calculus, and then project surface genericity theorem accordingly.

Adapting meaningfulness from dCBN to dCBV by replacing dCBN-reduction with dCBV-reduction may seem initially promising. This notion, known as dCBV-solvability, has appealing properties [64, 71, 11, 30, 47, 6]. Unfortunately, Accattoli and Guerrieri showed that genericity fails in such setting [6], and that equating unsolvable terms yields an inconsistent theory (see *e.g.* [6]). Consequently, dCBV-meaningfulness cannot be identified with dCBV-solvability. Identifying appropriate notions to capture dCBV meaningful λ-terms and formally validating these notions has been a longstanding and challenging open question.

Paolini and Ronchi Della Rocca [64, 71] introduced the notion of *potentially valuability* for CBV, also studied in [63, 11, 30, 42] and renamed *(dCBV) scrutability* in [6]. This notion, which we introduce below, proves to be suitable dCBV-meaningfulness. Notably, it aligns seamlessly with dBang-meaningfulness through the dCBV-embedding and thus enjoys a genericity theorem.

▶ **Definition 28.** *A term $t \in \Lambda$ is **dCBV-meaningful** if there exists a testing context $\mathtt{T}_{\mathtt{V}}$ and a value v such that $\mathtt{T}_{\mathtt{V}}\langle t\rangle \to^*_{\mathrm{S}_{\mathtt{V}}} v$, where testing contexts are defined by $\mathtt{T}_{\mathtt{V}} ::= \diamond \mid \mathtt{T}_{\mathtt{V}}\, u \mid (\lambda x.\mathtt{T}_{\mathtt{V}})\, u$.*

For example $t = x(\lambda y.z)$ is dCBV-meaningful as $\mathtt{T}_{\mathtt{V}}\langle t\rangle \to^*_{\mathrm{S}_{\mathtt{V}}} \lambda y.z$ for $\mathtt{T}_{\mathtt{V}} = (\lambda x.\diamond)(\lambda z.z)$, while Ω and $x\Omega$ are dCBV-meaningless as for whatever testing context Ω and $x\Omega$ are plugged into, Ω will not be erased. Note that the set of testing contexts is the same as those of dCBN.

Notice that this definition closely mirrors that of dBang-meaningfulness, with the primary difference being the replacement of dBang values for those of dCBV. Since values are typed with multitypes, it is natural to take them as types of the observable terms in dCBV (*i.e.* $\mathcal{T}^{\mathtt{obs}}_{\mathcal{V}} := \{\mathcal{M} \mid \mathcal{M} \text{ multitype}\}$). Consequently, and thanks to the preservation of typing (Proposition 21.2), one easily shows that testability is preserved through the dCBV translation: if a term t is $\mathcal{V}$-testable, then its image $t^{\mathtt{v}}$ is $\mathcal{B}$-testable.

As in dCBN and unlike dBang, dCBV-meaningfulness can actually be characterized both *operationally*, through surface normalizability, and *logically*, through typability in system $\mathcal{V}$. Moreover, the logical characterization turns out to be equivalent to $\mathcal{V}$-testability, meaning that dCBV-meaningfulness is also characterized by means of typability and inhabitation.

▶ **Theorem 29** (Characterizations of dCBV-Meaningfulness [11, 6, 23])**.** *Let $t \in \Lambda$.*

1. ***(Operational)** t is dCBV-meaningful iff t is dCBV surface-normalizing.*
2. ***(Logical)** (1) t is dCBV-meaningful iff (2) t is $\mathcal{V}$-typable iff (3) t is $\mathcal{V}$-testable.*

The notion of observable aligns in dCBV and dBang, at least from the type perspective. This yields a simple fully semantical proof of the preservation of dCBV-meaningfulness.

▶ **Theorem 30.** *Let $t \in \Lambda$, then t is dCBV-meaningful iff $t^{\mathtt{v}}$ is dBang-meaningful.*

Proof.
($\Rightarrow$) We present here a semantical proof. Let t be dCBV-meaningful, then using Theorem 29, one has that t is $\mathcal{V}$-testable thus, by preservation of testability, $t^{\mathtt{v}}$ is $\mathcal{B}$-testable and one concludes that $t^{\mathtt{v}}$ is dBang-meaningful according to Theorem 7.
($\Leftarrow$) Let $t^{\mathtt{v}}$ be dBang-meaningful, then using Theorem 7, it is $\mathcal{B}$-testable thus $\mathcal{B}$-typable. By Proposition 21.2, t is $\mathcal{V}$-typable and thus t is dCBV-meaningful by Theorem 29. ◀

Observe for example that $\mathtt{I}$ and $\mathtt{I}^{\mathtt{v}} = !\mathtt{I}_!$ are both dCBV/dBang-meaningful while Ω and $\Omega^{\mathtt{v}} = \Omega_!$ are both dCBV/dBang-meaningless.

Theorem 30 states that dCBV-meaningfulness precisely aligns with dBang-meaningfulness on its image, strengthening the idea that these two notions are adequately chosen. Thanks to Theorem 30, we can now project surface genericity from dBang to dCBV.

▶ **Theorem 31** (dCBV Qualitative Surface Genericity)**.** *Let $\mathtt{F_V}$ be a full context. If $\mathtt{F_V}\langle t\rangle$ is dCBV-meaningful for some dCBV-meaningless $t \in \Lambda$, then $\mathtt{F_V}\langle u\rangle$ is dCBV-meaningful for every $u \in \Lambda$.*

Proof. Let $t \in \Lambda$ be dCBV-meaningless and $\mathtt{F_V}$ be a full context. Suppose that $\mathtt{F_V}\langle t\rangle$ is dCBV-meaningful, then using Theorem 30, $(\mathtt{F_V}\langle t\rangle)^{\mathtt{v}}$ is dBang-meaningful, and $t^{\mathtt{v}}$ is dBang-meaningless. By induction on $\mathtt{F_V}$, $(\mathtt{F_V}\langle t\rangle)^{\mathtt{v}} = \mathtt{F}^{\mathtt{v}}_{\mathtt{V}}\langle t^{\mathtt{v}}\rangle$ thus $\mathtt{F}^{\mathtt{v}}_{\mathtt{V}}\langle t^{\mathtt{v}}\rangle$ is dBang-meaningful. By Corollary 11, for any $u \in \Lambda$, $\mathtt{F}^{\mathtt{v}}_{\mathtt{V}}\langle u^{\mathtt{v}}\rangle$ is dBang-meaningful. So, by typing preservation (Proposition 21.2), $(\mathtt{F_V}\langle u\rangle)^{\mathtt{v}}$ is dBang-meaningful, and hence $\mathtt{F_V}\langle u\rangle$ is dCBV-meaningful using Theorem 30. ◀

We now discuss some crucial consequences of our previous results, captured by the use of $\lambda_{\mathtt{dCBV}}$-theories. A **$\lambda_{\mathtt{dCBV}}$-theory** is an equivalence $\equiv$ on Λ containing $\to_{\mathtt{F_V}}$ and closed under full contexts. Let $\mathcal{H}_{\mathtt{dCBV}}$ (also noted $\equiv_{\mathcal{H}_{\mathtt{dCBV}}}$) be the smallest $\lambda_{\mathtt{dCBV}}$-theory equating all dCBV-meaningless terms, and let $\mathcal{H}^*_{\mathtt{dCBV}}$ be defined as follows:

$$\mathcal{H}^*_{\mathtt{dCBV}} := \{(t,u) \mid \forall\, \mathtt{F_V} \text{ full context, } \mathtt{F_V}\langle t\rangle \text{ dCBV-meaningful} \Leftrightarrow \mathtt{F}\langle u\rangle \text{ dBang-meaningful}\}$$

As for dBang and dCBN, $\mathcal{H}^*_{\mathtt{dCBV}}$ is the maximal consistent $\lambda_{\mathtt{dCBV}}$-theory containing $\mathcal{H}_{\mathtt{dCBV}}$ and coincides with observational equivalence in dCBV (see [15]). Again, thanks to the preservation of meaningfulness via the dCBV-embedding $\cdot^{\mathtt{v}}$ (Theorem 30), we can relate the theories $\mathcal{H}_{\mathtt{dBang}}$ and $\mathcal{H}^*_{\mathtt{dBang}}$ (in dBang) to the corresponding ones in dCBV, that is, $\mathcal{H}_{\mathtt{dCBN}}$ and $\mathcal{H}^*_{\mathtt{dCBV}}$.

▶ **Theorem 32.** *Let $t, u \in \Lambda$.*
1. *If $t \equiv_{\mathcal{H}_{dCBV}} u$ then $t^{\mathtt{v}} \equiv_{\mathcal{H}_{dBang}} u^{\mathtt{v}}$.*
2. *If $t^{\mathtt{v}} \equiv_{\mathcal{H}^*_{dBang}} u^{\mathtt{v}}$ then $t \equiv_{\mathcal{H}^*_{dCBV}} u$.*

Proof.
1. Immediate consequence of Theorem 30 and Lemma 19.2
2. Let $t, u \in \Lambda$ such that $t^{\mathtt{v}} \equiv_{\mathcal{H}^*_{\mathtt{dBang}}} u^{\mathtt{v}}$. Let $\mathtt{F_V}$ be a full context and suppose that $\mathtt{F_V}\langle t\rangle$ is dCBV-meaningful. By Theorem 30, $(\mathtt{F_V}\langle t\rangle)^{\mathtt{v}}$ is dBang-meaningful, and by Theorem 7 $(\mathtt{F_V}\langle t\rangle)^{\mathtt{v}}$ is $\mathcal{B}$-testable. As $(\mathtt{F_V}\langle t\rangle)^{\mathtt{v}} \to^*_{\mathrm{F}} \mathtt{F}^{\mathtt{v}}_{\mathtt{V}}\langle t^{\mathtt{v}}\rangle$ (see [14]) and typing is preserved by reduction (Theorem 3.1), we deduce that $\mathtt{F}^{\mathtt{v}}_{\mathtt{V}}\langle t^{\mathtt{v}}\rangle$ is also dBang-meaningful. Since $t^{\mathtt{v}} \equiv_{\mathcal{H}^*_{\mathtt{dBang}}} u^{\mathtt{v}}$, one has that $\mathtt{F}^{\mathtt{v}}_{\mathtt{V}}\langle u^{\mathtt{v}}\rangle$ is dBang-meaningful and so is $(\mathtt{F_V}\langle u\rangle)^{\mathtt{v}}$, thanks to Theorem 3.1 and since $(\mathtt{F_V}\langle u\rangle)^{\mathtt{v}} \to^*_{\mathrm{F}} \mathtt{F}^{\mathtt{v}}_{\mathtt{V}}\langle u^{\mathtt{v}}\rangle$. By Theorem 30, $\mathtt{F_V}\langle u\rangle$ is dCBV-meaningful and hence $t \equiv_{\mathcal{H}^*_{\mathtt{dCBV}}} u$. ◀

As in the dCBN-case, we strongly conjecture that the converse of Theorem 32.1 also holds. As in the dCBN-case again, the converse of Theorem 32.2 is actually false. Indeed, the η_v-expansion is included in $\mathcal{H}^*_{\mathtt{dCBV}}$ (see [66, 53, 22, 3]) but not in $\mathcal{H}^*_{\mathtt{dBang}}$, *i.e.* $x \equiv_{\mathcal{H}^*_{\mathtt{dCBV}}} \lambda y.xy$ but $x^{\mathtt{v}} = !x \not\equiv_{\mathcal{H}^*_{\mathtt{dCBV}}} !\lambda y.x!y = (\lambda y.xy)^{\mathtt{v}}$: the context $\mathtt{F} = \mathtt{der}(\diamond)\,[x\backslash !!w]$ separates the two. Again, from the viewpoint of Theorem 15, this phenomenon is not so surprising as it tells us that the dCBV and dBang observational equivalence does not coincide on the image of $\cdot^{\mathtt{v}}$, since dBang is a finer language than dCBV, with more contexts to separate terms operationally.

6 Conclusion and Future Work

We defined a notion of meaningful term, in a unifying well-established framework dBang that is able to capture both dCBN and dCBV calculi. We validated this notion of meaningfulness by providing a (high-level) characterization based on both typability and inhabitation, and showing a (surface) genericity result. All these results in dBang are perfectly analogous to well-known results for dCBN and dCBV [15]. Furthermore, both meaningfulness and genericity in dBang are shown to capture their respective notions in dCBN and dCBV. This suggests that there is a sort of *canonicity* in our definition of dBang-meaningfulness.

It is natural to wonder why this work is not conducted on the usual CBN and CBV calculi but rather their distant version dCBN and dCBV, which make use of explicit substitutions. The main reason is the *non-adequacy* of Plotkin's CBV calculus [66], meaning that some observational equivalent terms have different operational behaviors. Indeed, take the term $t := (\lambda x.\Delta)\,(yy)\,\Delta$ which is observationally equivalent to the prototypical diverging term Ω. Since $\lambda x.\Delta$ is applied to yy – which is not a value and cannot reduce to a value – it makes t a normal form in Plotkin's CBV. This mismatch complicates the study of dCBV-meaningfulness. Notice that this issue is solved in dCBV as the term t now diverges: $t \to_{\mathrm{S_v}} \Delta[x\backslash yy]\Delta \to_{\mathrm{S_v}} (zz)[z\backslash\Delta][x\backslash yy] \to_{\mathrm{S_v}} \Omega[x\backslash yy] \to_{\mathrm{S_v}} \ldots$, as expected. Furthermore, the observational equivalences generated by Plotkin's CBV and dCBV coincide, making the calculus switch harmless. Since adequacy for CBV is recovered thanks to ES and action at a distance, it is then natural to adopt a similar specification for CBN, knowing that standard CBN λ-calculus and dCBN are operationally and semantically equivalent.

While the logical characterization of meaningfulness for dBang (Theorem 7) requires additional hypotheses (typability and inhabitation) compared to those for dCBN (Theorem 24) and dCBV (Theorem 29), which only require typability, this dissimilarity should not be mistakenly interpreted as a weakness of our approach.

Firstly, the inhabitation condition becomes trivial in the case of dCBN and dCBV, as testability and typability coincide in both cases. Consequently, our approach to meaningfulness for dBang clearly provides a conservative extension of those for dCBN and dCBV.

Secondly, the use of distinct term constructors to specify data that cannot be intermingled seems unavoidable to embed both call-by-name and call-by-value calling paradigms within a single unifying framework. In the case of dBang, a clear distinction must be made between functions (represented by abstractions) and duplicable terms (represented by bang). This syntactic distinction, absent in both call-by-name and call-by-value, results in a unifying framework containing (at least) two built-in primitives that capture *incompatible* data. Then, the use of intersection types enable, in principle, such mismatch to exists even though a term cannot actually be a bang and a function at the same time. To address this issue, it may seem tempting to explore some syntactical restriction of intersection type systems such as uniformity [65] or compatibility [29], but both these cases result in a loss of completeness.

Finally, the characterization of meaningfulness through typability and inhabitation in a language equipped with incompatible data structures was initially studied in [29, 26], in the context of a λ-calculus with pair patterns. Clearly, functions cannot be pattern-matched by pair patterns, and pairs cannot be applied to arguments.

Besides that, several questions remain to be explored. First of all, we aim to show that our notion of meaningfulness for dBang allows us to prove a *full* genericity result in dBang in Barendregt's sense as mentioned in Section 1 (meaningless subterms are computationally irrelevant in the evaluation of full normalizing terms). A notion of stratified reduction, a finer operational semantics generalizing surface reduction to different levels, has been recently defined for dCBN and dCBV [15]. Stratified reduction is a key tool to show a full genericity result for both dCBN and dCBV. We plan to transfer these techniques to the more general framework of dBang, so that full genericity for dCBN and dCBV can be simply obtained by projecting the more general notion of full genericity for dBang via CBN/CBV translations.

It has been observed [69] that dBang can be embedded in this pattern language. Nevertheless, these two languages are not semantically equivalent, as dBang allows only duplication of values (bang terms), whereas the pattern language allows duplication of arbitrary terms.

We also plan to further study the properties of the smallest theory $\mathcal{H}_{\mathtt{dBang}}$ generated by equating all the meaningful terms in dBang. We strongly conjecture that $\mathcal{H}_{\mathtt{dBang}}$ restricted to the image of the embedding $\cdot^{\mathtt{n}}$ (resp. $\cdot^{\mathtt{v}}$) is equivalent to $\mathcal{H}_{\mathtt{dCBN}}$ in dCBN (resp. $\mathcal{H}_{\mathtt{dCBV}}$ in dCBV).

We would like to extend our study to other natural objects in the theory of programming, such as Böhm trees for dBang and their related theorems (*e.g.* approximation and separability). Böhm trees for dBang are expected to encompass both dCBN [20] and dCBV [49] ones.

Unifying frameworks such as dBang should also provide other general results for dCBN and dCBV, such as standardization, separability, etc. All this is left to future work. Finally, a more ambitious goal would be to generalize these results to models of computations with effects, such as global memory, non-determinism, exceptions, etc. This would approach our study on dBang to a more general unifying framework such as call-by-push-value [54, 55].

References

1 Beniamino Accattoli. An abstract factorization theorem for explicit substitutions. In Ashish Tiwari, editor, *23rd International Conference on Rewriting Techniques and Applications (RTA'12) , RTA 2012, May 28 - June 2, 2012, Nagoya, Japan*, volume 15 of *LIPIcs*, pages 6–21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPICS.RTA.2012.6.

2 Beniamino Accattoli, Eduardo Bonelli, Delia Kesner, and Carlos Lombardi. A nonstandard standardization theorem. In Suresh Jagannathan and Peter Sewell, editors, *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 659–670. ACM, 2014. doi:10.1145/2535838.2535886.

3 Beniamino Accattoli, Claudia Faggian, and Adrienne Lancelot. Normal form bisimulations by value. *CoRR*, abs/2303.08161, 2023. doi:10.48550/arXiv.2303.08161.

4 Beniamino Accattoli, Stéphane Graham-Lengrand, and Delia Kesner. Tight typings and split bounds, fully developed. *Journal of Functional Programming*, 30:e14, 2020. doi:10.1017/S095679682000012X.

5 Beniamino Accattoli and Giulio Guerrieri. Call-by-value solvability and multi types. *CoRR*, abs/2202.03079, 2022. arXiv:2202.03079.

6 Beniamino Accattoli and Giulio Guerrieri. The theory of call-by-value solvability. *Proceedings of the ACM on Programming Languages*, 6(ICFP):855–885, 2022. doi:10.1145/3547652.

7 Beniamino Accattoli and Delia Kesner. The structural *lambda*-calculus. In *Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL. Proceedings*, volume 6247 of *Lecture Notes in Computer Science*, pages 381–395. Springer, 2010. doi:10.1007/978-3-642-15205-4_30.

8 Beniamino Accattoli and Delia Kesner. The permutative λ-calculus. In Nikolaj S. Bjørner and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 18th International Conference, LPAR-18, Mérida, Venezuela, March 11-15, 2012. Proceedings*, volume 7180 of *Lecture Notes in Computer Science*, pages 23–36. Springer, 2012. doi:10.1007/978-3-642-28717-6_5.

9 Beniamino Accattoli and Delia Kesner. Preservation of strong normalisation modulo permutations for the structural lambda-calculus. *Log. Methods Comput. Sci.*, 8(1), 2012. doi:10.2168/LMCS-8(1:28)2012.

10 Beniamino Accattoli and Adrienne Lancelot. Light genericity. In Naoki Kobayashi and James Worrell, editors, *Foundations of Software Science and Computation Structures - 27th International Conference, FoSSaCS 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6-11, 2024, Proceedings, Part II*, volume 14575 of *Lecture Notes in Computer Science*, pages 24–46. Springer, 2024. doi:10.1007/978-3-031-57231-9_2.

11 Beniamino Accattoli and Luca Paolini. Call-by-value solvability, revisited. In *Functional and Logic Programming - 11th International Symposium, FLOPS 2012. Proceedings*, volume 7294 of *Lecture Notes in Computer Science*, pages 4–16. Springer, 2012. doi:10.1007/978-3-642-29822-6_4.

12 Sandra Alves, Delia Kesner, and Daniel Ventura. A quantitative understanding of pattern matching, 2019. arXiv:1912.01914.

13 Victor Arrial, Giulio Guerrieri, and Delia Kesner. Quantitative Inhabitation for Different Lambda Calculi in a Unifying Framework. *Proceedings of the ACM on Programming Languages*, 7(POPL):51:1483–51:1513, January 2023. doi:10.1145/3571244.

14 Victor Arrial, Giulio Guerrieri, and Delia Kesner. The benefits of diligence. In Chris Benzmüller, Marijn Heule, and Renate Schmidt, editors, *12th International Joint Conference on Automated Reasoning (IJCAR), 2024, Proceedings*, Lecture Notes in Artificial Intelligence. Springer, 2024.

15 Victor Arrial, Giulio Guerrieri, and Delia Kesner. Genericity through stratification. *CoRR*, abs/2401.12212, 2024. doi:10.48550/arXiv.2401.12212.

16 Davide Barbarossa and Giulio Manzonetto. Taylor subsumes scott, berry, kahn and plotkin. *Proc. ACM Program. Lang.*, 4(POPL):1:1–1:23, 2020. doi:10.1145/3371069.

17 Henk Barendregt. *Some extensional term models for combinatory logics and λ-calculi*. PhD thesis, Univ. Utrecht, January 1971.

18 Henk Barendregt. A characterization of terms of the lambda i-calculus having a normal form. *Journal Symbolic Logic*, 38(3):441–445, 1973. doi:10.2307/2273041.

19 Henk Barendregt. Solvability in lambda-calculi. In M. Guillaume, editor, *Colloque international de logique: Clermont-Ferrand, 18-25 juillet 1975*, pages 209–219, Paris, 1977. Éditions du CNRS.

20 Henk Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in logic and the foundation of mathematics*. North-Holland, Amsterdam, revised edition, 1984.

21 Jan Bessai, Tzu-Chun Chen, Andrej Dudenhefner, Boris Düdder, Ugo de'Liguoro, and Jakob Rehof. Mixin composition synthesis based on intersection types. *CoRR*, abs/1712.06906, 2017. arXiv:1712.06906.

22 Dariusz Biernacki, Sergueï Lenglet, and Piotr Polesiuk. Proving soundness of extensional normal-form bisimilarities. *Log. Methods Comput. Sci.*, 15(1), 2019. doi:10.23638/LMCS-15(1:31)2019.

23 Antonio Bucciarelli, Delia Kesner, Alejandro Ríos, and Andrés Viso. The bang calculus revisited. In *Functional and Logic Programming - 15th International Symposium, FLOPS 2020, Proceedings*, volume 12073 of *Lecture Notes in Computer Science*, pages 13–32. Springer, 2020. doi:10.1007/978-3-030-59025-3_2.

24 Antonio Bucciarelli, Delia Kesner, Alejandro Ríos, and Andrés Viso. The bang calculus revisited. *Inf. Comput.*, 293:105047, 2023. doi:10.1016/J.IC.2023.105047.

25 Antonio Bucciarelli, Delia Kesner, and Simona Ronchi Della Rocca. The inhabitation problem for non-idempotent intersection types. In Josep Díaz, Ivan Lanese, and Davide Sangiorgi, editors, *Theoretical Computer Science - 8th IFIP TC 1/WG 2.2 International Conference, TCS 2014, Rome, Italy, September 1-3, 2014. Proceedings*, volume 8705 of *Lecture Notes in Computer Science*, pages 341–354. Springer, 2014. doi:10.1007/978-3-662-44602-7_26.

26 Antonio Bucciarelli, Delia Kesner, and Simona Ronchi Della Rocca. Solvability = Typability + Inhabitation. *Logical Methods in Computer Science*, Volume 17, Issue 1, January 2021. doi:10.23638/LMCS-17(1:7)2021.

27 Antonio Bucciarelli, Delia Kesner, and Simona Ronchi Della Rocca. Solvability = typability + inhabitation. *Logical Methods in Computer Science*, 17(1), 2021. URL: https://lmcs.episciences.org/7141.

28 Antonio Bucciarelli, Delia Kesner, and Simona Ronchi Della Rocca. Inhabitation for non-idempotent intersection types. *Logical Methods in Computer Science*, 14(3), 2018. doi:10.23638/LMCS-14(3:7)2018.

29 Antonio Bucciarelli, Delia Kesner, and Simona Ronchi Della Rocca. Observability for pair pattern calculi. In *TLCA*, volume 38 of *LIPIcs*, pages 123–137. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, 2015.

30 Alberto Carraro and Giulio Guerrieri. A semantical and operational account of call-by-value solvability. In Anca Muscholl, editor, *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, Proceedings*, volume 8412 of *Lecture Notes in Computer Science*, pages 103–118, Grenoble, France, 2014. Springer. doi:10.1007/978-3-642-54830-7_7.

31 Mario Coppo and Mariangiola Dezani-Ciancaglini. A new type assignment for λ-terms. *Archiv für mathematische Logik und Grundlagenforschung*, 19(1):139–156, 1978. doi:10.1007/BF02011875.

32 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. doi:10.1305/ndjfl/1093883253.

33 Mario Coppo, Mariangiola Dezani-Ciancaglini, and Betti Venneri. Functional characters of solvable terms. *Mathematical Logic Quarterly*, 27(2-6):45–58, 1981. doi:10.1002/malq.19810270205.

34 Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In Martin Odersky and Philip Wadler, editors, *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00)*, pages 233–243. ACM, 2000. doi:10.1145/351240.351262.

35 Pierre-Louis Curien and Guillaume Munch-Maccagnoni. The duality of computation under focus. In Cristian S. Calude and Vladimiro Sassone, editors, *Theoretical Computer Science - 6th IFIP TC 1/WG 2.2 International Conference, TCS 2010, Held as Part of WCC 2010. Proceedings*, volume 323 of *IFIP Advances in Information and Communication Technology*, pages 165–181. Springer, 2010. doi:10.1007/978-3-642-15240-5_13.

36 Daniel de Carvalho. *Sémantiques de la logique linéaire et temps de calcul.* PhD thesis, Université Aix-Marseille II, 2007.

37 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.

38 Thomas Ehrhard. Call-by-push-value from a linear logic point of view. In Peter Thiemann, editor, *Programming Languages and Systems - 25th European Symposium on Programming, ESOP 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Proceedings*, volume 9632 of *Lecture Notes in Computer Science*, pages 202–228, Eindhoven, The Netherlands, 2016. Springer. doi:10.1007/978-3-662-49498-1_9.

39 Thomas Ehrhard and Giulio Guerrieri. The bang calculus: an untyped lambda-calculus generalizing call-by-name and call-by-value. In James Cheney and Germán Vidal, editors, *Proceedings of the 18th International Symposium on Principles and Practice of Declarative Programming*, pages 174–187, Edinburgh, United Kingdom, 2016. ACM. doi:10.1145/2967973.2968608.

40 José Espírito Santo, Luís Pinto, and Tarmo Uustalu. Modal embeddings and calling paradigms. In *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019*, volume 131 of *LIPIcs*, pages 18:1–18:20. Schloss Dagstuhl, 2019. doi:10.4230/LIPIcs.FSCD.2019.18.

41 Claudia Faggian and Giulio Guerrieri. Factorization in call-by-name and call-by-value calculi via linear logic. In *Foundations of Software Science and Computation Structures - 24th International Conference, FOSSACS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Proceedings*, volume 12650 of *Lecture Notes in Computer Science*, pages 205–225. Springer, 2021. doi:10.1007/978-3-030-71995-1_11.

42 Álvaro García-Pérez and Pablo Nogueira. No solvable lambda-value term left behind. *Logical Methods in Computer Science*, 12(2):1–43, 2016. doi:10.2168/LMCS-12(2:12)2016.

43 Philippa Gardner. Discovering needed reductions using type theory. In Masami Hagiya and John C. Mitchell, editors, *Theoretical Aspects of Computer Software*, pages 555–574, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

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

45 Jean-Yves Girard. On the unity of logic. *Ann. Pure Appl. Log.*, 59(3):201–217, 1993. doi:10.1016/0168-0072(93)90093-S.

46 Giulio Guerrieri and Giulio Manzonetto. The bang calculus and the two girard's translations. In *Proceedings Joint International Workshop on Linearity & Trends in Linear Logic and Applications, Linearity-TLLA@FLoC 2018*, volume 292 of *EPTCS*, pages 15–30, Oxford, UK, 2018. doi:10.4204/EPTCS.292.2.

47 Giulio Guerrieri, Luca Paolini, and Simona Ronchi Della Rocca. Standardization and conservativity of a refined call-by-value lambda-calculus. *Logical Methods Computer Science*, 13(4):1–27, 2017. doi:10.23638/LMCS-13(4:29)2017.

48 Richard Kennaway, Vincent van Oostrom, and Fer-Jan de Vries. Meaningless terms in rewriting. *J. Funct. Log. Program.*, 1999(1), 1999. URL: http://danae.uni-muenster.de/lehre/kuchen/JFLP/articles/1999/A99-01/A99-01.html.

49 Axel Kerinec, Giulio Manzonetto, and Michele Pagani. Revisiting call-by-value böhm trees in light of their taylor expansion. *Logical Methods in Computer Science*, 16(3):6:1–6:26, 2020. URL: https://lmcs.episciences.org/6638.

50 Delia Kesner, Victor Arrial, and Giulio Guerrieri. Meaningfulness and genericity in a subsuming framework. *CoRR*, abs/2404.06361, 2024. URL: https://arxiv.org/abs/2404.06361.

51 Delia Kesner and Shane Ó Conchúir. Milner's lambda-calculus with partial substitutions. *CoRR*, abs/2312.13270, 2023. doi:10.48550/arXiv.2312.13270.

52 Delia Kesner and Andrés Viso. Encoding tight typing in a unified framework. In *30th EACSL Annual Conference on Computer Science Logic, CSL 2022*, volume 216 of *LIPIcs*, pages 27:1–27:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.CSL.2022.27.

53 Søren B. Lassen. Eager normal form bisimulation. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005), 26-29 June 2005, Chicago, IL, USA, Proceedings*, pages 345–354. IEEE Computer Society, 2005. doi:10.1109/LICS.2005.15.

54 Paul Blain Levy. Call-by-push-value: A subsuming paradigm. In Jean-Yves Girard, editor, *Typed Lambda Calculi and Applications*, pages 228–243, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg. doi:10.1007/3-540-48959-2_17.

55 Paul Blain Levy. *Call-By-Push-Value: A Functional/Imperative Synthesis*, volume 2 of *Semantics Structures in Computation*. Springer, 2004.

56 Zohar Manna and Richard J. Waldinger. A deductive approach to program synthesis. *ACM Transactions on Programming Languages and Systems*, 2(1):90–121, 1980. doi:10.1145/357084.357090.

57 John Maraist, Martin Odersky, David N. Turner, and Philip Wadler. Call-by-name, call-by-value, call-by-need and the linear lambda calculus. In Stephen D. Brookes, Michael G. Main, Austin Melton, and Michael W. Mislove, editors, *Eleventh Annual Conference on Mathematical Foundations of Programming Semantics, MFPS 1995, Tulane University, New Orleans, LA, USA, March 29 - April 1, 1995*, volume 1 of *Electronic Notes in Theoretical Computer Science*, pages 370–392. Elsevier, 1995. doi:10.1016/S1571-0661(04)00022-2.

58 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.

59 Robin Milner. Local bigraphs and confluence: two conjectures. In Roberto Amadio and Iain Phillips, editors, *Proceedings of the 13th Int. Workshop on Expressiveness in Concurrency (EXPRESS)*, volume 175, pages 65–73. Electronic Notes in Theoretical Computer Science, 2006. doi:10.1016/j.entcs.2006.07.035.

60 Eugenio Moggi. Computational lambda-calculus and monads. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89), Pacific Grove, California, USA, June 5-8, 1989*, pages 14–23. IEEE Computer Society, 1989. doi:10.1109/LICS.1989.39155.

61 Eugenio Moggi. Notions of computation and monads. *Inf. Comput.*, 93(1):55–92, 1991. doi:10.1016/0890-5401(91)90052-4.

62 Guillaume Munch-Maccagnoni. Focalisation and classical realisability. In *Computer Science Logic, 23rd international Workshop, CSL 2009, 18th Annual Conference of the EACSL. Proceedings*, volume 5771 of *Lecture Notes in Computer Science*, pages 409–423. Springer, 2009. doi:10.1007/978-3-642-04027-6_30.

63 Luca Paolini, Elaine Pimentel, and Simona Ronchi Della Rocca. An operational characterization of strong normalization. In Luca Aceto and Anna Ingólfsdóttir, editors, *Foundations of Software Science and Computation Structures, 9th International Conference, FOSSACS 2006, Proceedings*, volume 3921 of *Lecture Notes in Computer Science*, pages 367–381, Vienna, Austria, 2006. Springer. doi:10.1007/11690634_25.

64 Luca Paolini and Simona Ronchi Della Rocca. Call-by-value solvability. *RAIRO Theoretical Informatics and Applications*, 33(6):507–534, 1999. doi:10.1051/ita:1999130.

65 Daniele Pautasso and Simona Ronchi Della Rocca. A quantitative version of simple types. In Marco Gaboardi and Femke van Raamsdonk, editors, *8th International Conference on Formal Structures for Computation and Deduction, FSCD 2023, July 3-6, 2023, Rome, Italy*, volume 260 of *LIPIcs*, pages 29:1–29:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.FSCD.2023.29.

66 Gordon D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theoretical Computer Science*, 1(2):125–159, 1975. doi:10.1016/0304-3975(75)90017-1.

67 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.

68 Dag Prawitz. *Classical versus intuitionistic logic. Why is this a Proof?*, volume 27. College Publications, 2017.

69 Miguel Ramos. Embedding the distant bang calculus with pattern-matching primitives, 2024. Personal Communication.

70 Laurent Regnier. Une équivalence sur les lambda-termes. *Theoretical Computer Science*, 126(2):281–292, 1994. doi:10.1016/0304-3975(94)90012-4.

71 Simona Ronchi Della Rocca and Luca Paolini. *The Parametric Lambda Calculus - A Metamodel for Computation*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004. doi:10.1007/978-3-662-10394-4.

72 Simona Ronchi Della Rocca and Luca Roversi. Lambda calculus and intuitionistic linear logic. *Stud Logica*, 59(3):417–448, 1997. doi:10.1023/A:1005092630115.

73 Nicolas Wu Steffen van Bakel, Emma Tye. A calculus of delayed reductions. In *PPDP 2023: 25th International Symposium on Principles and Practice of Declarative Programming, Cascais, Lisbon, Portugal, October 22 - 23, 2022*. ACM, 2023.

74 Pawel Urzyczyn. The emptiness problem for intersection types. *Journal of Symbolic Logic*, 64(3):1195–1215, 1999. doi:10.2307/2586625.

75 Christopher P. Wadsworth. *Semantics and pragmatics of the lambda-calculus*. Ph.D., University of Oxford, 1971. Accepted: 1971. URL: https://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.476186.

76 Christopher P. Wadsworth. The Relation between Computational and Denotational Properties for Scott's D_∞-Models of the Lambda-Calculus. *SIAM Journal on Computing*, 5(3):488–521, September 1976. Publisher: Society for Industrial and Applied Mathematics. doi:10.1137/0205036.

Abstraction-Based Decision Making for Statistical Properties

Filip Cano
Graz University of Technology, Austria

Thomas A. Henzinger
Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Bettina Könighofer
Graz University of Technology, Austria

Konstantin Kueffner
Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Kaushik Mallik
Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Abstract

Sequential decision-making in probabilistic environments is a fundamental problem with many applications in AI and economics. In this paper, we present an algorithm for synthesizing sequential decision-making agents that optimize *statistical properties* such as maximum and average response times. In the general setting of sequential decision-making, the environment is modeled as a random process that generates inputs. The agent responds to each input, aiming to maximize rewards and minimize costs within a specified time horizon. The corresponding synthesis problem is known to be PSPACE-hard. We consider the special case where the input distribution, reward, and cost depend on input-output statistics specified by counter automata. For such problems, this paper presents the first PTIME synthesis algorithms. We introduce the notion of *statistical abstraction*, which clusters statistically indistinguishable input-output sequences into equivalence classes. This abstraction allows for a dynamic programming algorithm whose complexity grows polynomially with the considered horizon, making the *statistical case* exponentially more efficient than the general case. We evaluate our algorithm on three different application scenarios of a client-server protocol, where multiple clients compete via bidding to gain access to the service offered by the server. The synthesized policies optimize profit while guaranteeing that none of the server's clients is disproportionately starved of the service.

2012 ACM Subject Classification Theory of computation → Online algorithms; Theory of computation → Computational pricing and auctions; Theory of computation → Abstraction

Keywords and phrases Abstract interpretation, Sequential decision making, Counter machines

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.2

Category Invited Talk

Supplementary Material
Software: `https://github.com/filipcano/abstraction-based-decision-making-FSCD24`
archived at `swh:1:dir:1113fa6039e8d3792e47a1fed24cc5210159ebef`

Funding This work is partly supported by the European Research Council under Grant No.: ERC-2020-AdG 101020093. It is also partially supported by the State Government of Styria, Austria – Department Zukunftsfonds Steiermark.

1 Introduction

Sequential decision-making is a core algorithmic task in many AI-based planning and control problems with uncertain environments. The environment produces a sequence of inputs one at a time, and the decision-making agent needs to produce outputs on each of the inputs as soon as they arrive – without having seen the inputs from the future. We consider the finite-horizon setting, where the decision horizon, i.e., the total number of inputs to appear, is finite and fixed apriori.

The problem of computing an optimal decision-making agent in the sequential setting is known to be PSPACE-hard. Our contribution is to introduce a class of statistical properties for which the same problem can be solved in polynomial time. A property is called statistical if its satisfaction can be measured by keeping track of small statistics, typically a fixed set of states and a fixed set of counter values. The class of statistical properties not only includes every regular property but also includes richer quantitative properties like maximum and average response times. We present a new synthesis algorithm for sequential decision-makers whose constraint, objective function, and environment model can all be captured using statistical properties. The complexity of our algorithm, which is based on a notion of *statistical abstraction*, grows only polynomially with respect to the considered horizon.

The problem setting. The inputs to our synthesis algorithm are a probabilistic model of the environment generating a random sequence of inputs, a horizon length, a qualitative constraint function, and a quantitative objective function over the generated input-output sequence. The environment model, the constraint, and the objective function are required to be given as statistical properties, which guarantee that their outputs depend only on some small statistics about the past sequence of inputs and outputs. Since our statistical properties include regular properties, the qualitative constraint can be any finite-state constraint. Henceforth, we will refer to the constraint and the objective function as *cost constraint* and *reward*, respectively. We propose a synthesis procedure to compute a decision maker that, at each step, reads the current environment input and computes an output such that the expected reward is maximized over all possible futures that satisfy the cost constraint up to the given horizon. If no such future is possible, the decision maker outputs "fail."

Example – Synthesis of responsive servers. We consider a simple client-server model, inspired by online advertisements [10], where multiple clients (i.e., the advertisers) compete via a bidding mechanism to access a certain service (i.e., the act of putting up the ads) hosted by the server (i.e., the advertisement publisher). A greedy server would always accept the highest bidder to maximize its profit, but this could potentially starve the weaker clients, monopolize the market, and ultimately result in overall bad ratings that are harmful to its business in the long run. We consider three different synthesis problems for designing servers that strike a balance between profit-making and maintaining acceptable service quality for all clients – both aspects being representable as statistical properties.

Synthesis problem I: Balanced servers. We say that a server is balanced if the individual frequencies of the clients getting access are similar. A *balanced server* ensures that no client experiences a significantly lower number of server accesses compared to the other clients. Given a time horizon t, a constant probability distribution over the prices that the clients will offer, and given a bound on the "imbalance," how do we compute a server policy that will maximize the expected profit while keeping the imbalance within bounds in time t?

Synthesis problem II: Maximally responsive servers. While the balanced server guarantees that each client would get a similar *total* number of accesses by the end of the horizon, the clients may end up waiting for longer periods. *Maximally responsive servers* minimize the maximal waiting time for each client. As a trade-off, the server may need to occasionally select the client offering a lower price. We quantify this trade-off using the *opportunity cost*, defined as the extra profit the server could make by selecting the client offering the highest price at a given stage. Given a time horizon t, a constant distribution over prices that the clients will offer, and a bound k on the total opportunity costs, how do we compute a server policy that is maximally responsive while making sure that the sum of opportunity costs after time t does not exceed k?

Synthesis problem III: Clientele-aware servers. So far, the probability distribution over the offer prices by the clients was assumed to remain constant. The more realistic setting is when the clients adjust their prices based on their past experiences with the service: a dissatisfied client would lower the offer prices in the future whereas a satisfied client would be willing to pay more. Therefore, a far-sighted server policy should limit its initial rejection rates to keep up the expected offer prices in the future – even if this may need sacrificing the profit in the beginning. We consider the setting with just one client and assume that the server has a budget of N on the number of times the client can be accepted. Given a time horizon t and a model of how the past decisions affect the future price distributions, how do we compute a server policy that will maximize the expected profit while accepting the client's request for at most N times?

Algorithmic solution and complexity. We present synthesis algorithms for the case that the probability distribution of the environment, the reward, and the cost constraint are all provided as statistical properties. For the general class of properties, the optimal decision of the agent at any point may need to consider every possible sequence of future inputs and outputs. This causes an expensive blow-up that is unavoidable in general, as the problem is known to be PSPACE-hard [12].

Our key insight is that for properties we call *statistical*, such as the maximum response time of a server, the computational blow-up can be avoided by using a statistical abstraction of the history. Such an abstraction of the observed input-output sequences combines statistically indistinguishable sequences into the same equivalence classes. For properties for which the number of equivalence classes grows polynomially with time, we obtain a *polynomial-time algorithm* for computing the optimal decision maker. Any property that can be described using a constant number of counters (like the maximum response time in the aforementioned client-server examples) and a constant number of states (like regular languages) falls into this category. In contrast, any order-dependent property (like a discounted sum) would have an exponentially growing number of equivalence classes. Note, that there are other common "statistics," such as variance, which are more complicated to compute and do not lie within our class of statistical properties.

We show how to adapt the standard dynamic programming algorithm to work on the abstracted domain, rather than on the full history. We also show a correspondence between the size of the abstraction and the size of the representation of statistical properties using *counter automata*.

Experiments. We evaluated our method on the three examples described above. For each example, we compared the computational performances of the dynamic programming algorithm with and without using a statistical abstraction. Without the abstraction, the

algorithm runs out of memory (with a memory limit of 200 GB) for horizon lengths less than 10 in all cases. With the abstraction, however, the algorithm manages much larger instances, exceeding 400 in all cases and reaching up to 60000 for the balanced server.

Furthermore, we demonstrate how our decision-makers fare in simulations. For balanced servers, we demonstrate that our synthesized policy maintains balance while achieving performance similar to that of a greedy policy. Additionally, it significantly outperforms a policy synthesized to be balanced on expectation. For maximally responsive servers, we show how our method produces policies that can significantly reduce maximum waiting time, paying a comparatively small price in opportunity cost. Finally, we analyse the policies synthesized for clientele-aware servers for different budgets.

2 Sequential Decision Making for Statistical Properties

2.1 The Sequential Decision Making Problem

We first formulate a general problem statement for sequential decision making involving arbitrary *quantitative* properties. Afterwards, we will identify the important subclass of the problem where all the quantitative properties are actually statistical in nature, like the ones we described in Sec. 1.

We consider the alternating turn-based interaction of the decision maker with its uncertain environment. At each interaction phase, called a *stage*, the environment samples an action from a known probability distribution over the *input alphabet* $\mathcal{X}$, and the decision maker responds by generating an action from the *output alphabet* $\mathcal{Y}$, and both actions may depend on the past stages. Formally, the environment is modeled as *stochastic generators* of the form $\theta\colon (\mathcal{X}\times\mathcal{Y})^* \to \Delta(\mathcal{X})$, and the system is modeled as *transducers* of the form $\gamma\colon (\mathcal{X}\times\mathcal{Y})^*\times\mathcal{X}\to\mathcal{Y}$. Let Θ be the set of every stochastic generator and Γ be the set of every transducer. The interaction between θ and γ induces a probability distribution $\mathbb{P}_{\theta,\gamma}$ over the space of finite input-output sequences – called *traces* – as follows. For every stage $t>0$ and for every trace $\vec{z}_t = z_1\dots z_t \in (\mathcal{X}\times\mathcal{Y})^t$,

$$\mathbb{P}_{\theta,\gamma}(\vec{z}_t) \coloneqq \prod_{i=1}^{t} \mathbb{P}_{\theta,\gamma}(z_i \mid \vec{z}_{i-1}),$$

where

$$\mathbb{P}_{\theta,\gamma}(z_i \mid \vec{z}_{i-1}) \coloneqq \begin{cases} \theta(\vec{z}_{i-1})(x_i) & \text{if } \gamma(\vec{z}_{i-1},x_i)=y_i,\\ 0 & \text{otherwise,}\end{cases}$$

with the convention that $\vec{z}_0$ represents the empty word. We will write $Dom(\mathbb{P}_{\theta,\gamma})$ to denote the domain of $\mathbb{P}_{\theta,\gamma}$, i.e., the set of traces with positive probabilities. Sometimes, we will call traces as *histories* to stress that the given input-output sequence consists of concrete inputs and outputs as observed in the past.

We consider a lexicographic specification for the transducer, formalized as follows. We consider the real-valued *reward function* $\texttt{rew}\colon (\mathcal{X}\times\mathcal{Y})^*\to\mathbb{R}$, and the binary *cost function* $\texttt{cost}\colon (\mathcal{X}\times\mathcal{Y})^*\to\{0,1\}$ (where "1" could represent that the cost is below a given threshold). We define the expected value of the reward after stage t as $\mathbb{E}^t_{\theta,\gamma}[\texttt{rew}] \coloneqq \sum_{\vec{z}_t\in(\mathcal{X}\times\mathcal{Y})^t} \texttt{rew}(\vec{z}_t)\cdot\mathbb{P}_{\theta,\gamma}(\vec{z}_t)$. The lexicographic specification requires the cost to be 1 for every probable input sequence of the environment ("hard" objective) and the expected reward is maximized ("soft" objective), at the end of a given number of stages.

▶ **Problem 1.** Let $T \in \mathbb{N}$ be the given horizon length (i.e., the last stage index), θ be the given stochastic generator, `rew` be the given reward function, and `cost` be the given cost function. Let $\Gamma^{\theta}_{\texttt{feas}} \subseteq \Gamma$ be the set of feasible transducers fulfilling the cost constraint in the presence of θ, i.e.,

$$\Gamma^{\theta}_{\texttt{feas}} \coloneqq \{\gamma \in \Gamma \mid \forall \vec{z}_T \in \mathit{Dom}(\mathbb{P}_{\theta,\gamma}) \,.\, \texttt{cost}(\vec{z}_T) = 1\}. \tag{1}$$

Compute the *optimal* transducer γ^* that is feasible and is reward-optimal, i.e.,

$$\gamma^* = \arg\max_{\gamma \in \Gamma^{\theta}_{\texttt{feas}}} \mathbb{E}^{T}_{\theta,\gamma}[\texttt{rew}]\,. \tag{2}$$

We will call the tuple $\langle \mathcal{X}, \mathcal{Y}, \theta, \texttt{rew}, \texttt{cost}, T\rangle$ a *problem instance*.

The finiteness of the horizon length is a standard choice in the literature on sequential decision making problems [9, 2] and is a natural choice in many practical situations. Naïvely lifting this finite-horizon restriction is technically tricky because it is unclear how $\Gamma^{\theta}_{\texttt{feas}}$ will be defined in the first place.

It follows from known results [12] that Prob. 1 is PSPACE-hard in general; see Sec. 3.1 for details. We present a PTIME algorithm for the special case where the functions θ, `rew`, and `cost` depend only on some particular statistics of histories.

2.2 Statistical Properties

In many practical instances of Prob. 1, including the motivating examples in Sec. 1, the functions `rew`, `cost`, and θ depend only on some aggregated statistic of the history, and the exact order of the inputs and outputs is unimportant. We formalize this in the following.

▶ **Definition 2** (Statistics). *Let $\mathcal{W}$ be an alphabet and $\mathcal{S}$ be an output domain. An $\mathcal{S}$-statistic over $\mathcal{W}$ is a function $\mu\colon \mathcal{W}^* \to \mathcal{S}$.*

We will omit $\mathcal{W}$ or $\mathcal{S}$ whenever they are either clear from the context or are unimportant. Following is the key property of statistics that will be useful to us.

▶ **Definition 3** (Well-behaved statistics). *A statistic μ over $\mathcal{W}$ is called well-behaved if for every $\vec{u}, \vec{v}, \vec{w} \in \mathcal{W}^*$, $\mu(\vec{u}) = \mu(\vec{v})$ implies $\mu(\vec{u}\vec{w}) = \mu(\vec{v}\vec{w})$.*

These concepts are illustrated using the following example.

▶ **Example 4.** Suppose we are given a sequence of the toss outcomes of a given coin, where heads and tails are represented as "1" and "0," respectively. Following are examples of statistics over $\{0, 1\}$: total number of heads $\mu_1(\cdot)$, largest number of consecutive tails between any two heads $\mu_2(\cdot)$, average number of heads $\mu_3(\cdot)$, the mode statistic $\mu_4(\cdot)$, etc. Among these, it can be easily verified that μ_1 and μ_2 are well-behaved. The average statistic μ_3 is not well-behaved: The sequences 10 and 1010 have the same averages (which is $1/2$), but extending them with the sequence 11 gives us different averages ($3/4$ and $2/3$, respectively). The mode statistic μ_4 is also not well-behaved: the sequences 110 and 1110 have the same mode 1, but extending them with the sequence 00 gives us different modes (0 and 1, respectively).

Let μ be an $\mathcal{S}$-statistic over the alphabet $\mathcal{W}$. For every $t > 0$, the statistic μ induces an equivalence relation $\equiv_{\mu,t}$ on the set $\mathcal{W}^t$, defined as follows: $\vec{w}_t \equiv_{\mu,t} \vec{z}_t$ iff $\mu(\vec{w}_t) = \mu(\vec{z}_t)$. For instance, for the statistic μ_1 from Ex. 4 and for $t = 3$, we have $110 \equiv_{\mu_1,3} 101 \equiv_{\mu_1,3} 011$, because $\mu_1(110) = \mu_1(101) = \mu_1(011) = 2$ and each of them have length 3.

In the subsequent sections, the equivalence relation $\equiv_{\mu,t}$ will give way to a small abstraction of the set $\mathcal{W}^t$, where all the words in a given equivalence class will be abstracted by a single representative word from that equivalence class. Therefore, the larger the number of equivalence classes, the larger and more complex will be the abstraction. We formalize this as a measure of the complexity of statistics.

▶ **Definition 5** (Size of statistics). *Let $\mu\colon \mathcal{W}^* \to \mathcal{S}$ be a statistic. The* size *of μ is the function $size_\mu\colon \mathbb{N} \to \mathbb{N}$ mapping every t to the number of equivalence classes in $\mathcal{W}^t$ induced by the equivalence relation $\equiv_{\mu,t}$.*

Consider the statistics μ_1 and μ_4 from Ex. 4. For any given t, it is easy to see that $size_{\mu_1}(t) = t+1$ (because there can be $0, 1, \ldots, t$ number of "1"-s) and $size_{\mu_4}(t) = 2$ (because the mode of any sequence can be either 0 or 1).

For a given $t \in \mathbb{N}$, let $\mathcal{S}_t \subseteq \mathcal{S}$ be the set of every valuation of μ on every word of length t, i.e., $\mathcal{S}_t = \mu(\mathcal{W}^t)$. A *$t$-reconstructor* of μ is any function $\kappa_t\colon \mathcal{S}_t \to \mathcal{W}^t$ such that if $\kappa_t(s) = \vec{w}_t$ then $\mu(\vec{w}_t) = s$. Observe that κ_t is not unique. For the statistic μ_1 in Ex. 4, we have $\mathcal{S}_t = \{0, 1, 2\}$, and one possible 2-reconstructor is given as $\kappa_2(0) = 00$, $\kappa_2(1) = 01$, and $\kappa_2(2) = 11$. The following claim follows immediately.

▶ **Proposition 6.** *Let $\mu\colon \mathcal{W} \to \mathcal{S}$ be a statistic and κ_t be a t-reconstructor of μ. Then for every $\vec{w} \in \mathcal{W}^t$, $\mu(\vec{w}) = \mu(\kappa_t \circ \mu(\vec{w}))$.*

We will now use well-behaved statistics to define the class of problem instances that are amenable to efficient computations and are our subject of study.

▶ **Definition 7** (μ-representability of functions). *Let μ be a $\mathcal{S}$-statistic over $\mathcal{W}$. The function $f\colon \mathcal{W}^* \to \mathcal{U}$ is μ-representable, if there exists a function $\widehat{f}\colon \mathcal{S} \to \mathcal{U}$ such that for every $\vec{w} \in \mathcal{W}^*$, $f(\vec{w}) = \widehat{f}(\mu(\vec{w}))$.*

We will later show that Prob. 1 can be efficiently solved if we can identify a well-behaved statistic μ such that θ, `rew`, and `cost` are μ-representible and μ is "small" in size. How to determine a small, well-behaved μ such that a given property f is μ-representible is a problem on its own. In Sec. 2.4, we present a pragmatic approach to quickly identify μ and its size when f is specified as a counter automaton. The problem of finding a small μ for f then boils down to the problem of finding a small counter automaton representing f.

2.3 Examples: Synthesis of Responsive Servers

Below, we describe how the examples from Sec. 1 can be formalized. We first introduce some common notation for the client-server model. Suppose that there are two clients, A and B, competing to access a resource hosted by a server. At each stage t, A and B concurrently submit their offer prices, a_t and b_t respectively, indicating the amount they are willing to pay for the service. Let a_t and b_t be non-negative integers, both less than or equal to C, where zero indicates that the respective client does not request the service at that stage. The server responds with the decision $d_t \in \mathcal{Y} = \{A, B\}$, representing whether client A or B get access. We model the pair of clients as the environment and the server as the system, i.e., $\mathcal{X} = [0; C] \times [0; C]$ and $\mathcal{Y} = \{A, B\}$.

▶ **Example 8** (Balanced server). Recall that the objective of the balanced server is to maximize profit while ensuring an even allocation of resources to its clients by the end of a given horizon. If the server initially acts greedily and accepts whoever offers the higher price, then it may later need to pay high opportunity costs (i.e., sacrifice profits to

balance out the past imbalances). We formulate the synthesis problem for the balanced server problem as follows. We assume that the distribution θ is known and remains fixed over time and that we have given an arbitrary trace $\vec{z}_t = (a_1, b_1)d_1 \dots (a_t, b_t)d_t \in (\mathcal{X} \times \mathcal{Y})^t$. We define the *profit* at stage i, denoted as $\texttt{Profit}(a_i, b_i, d_i)$, to be equal to a_i if $d_i = A$ and else equal to b_i (if $d_i = B$). Furthermore, we define the *imbalance* over $\vec{z}_t$, denoted as $\texttt{Imbalance}(\vec{z}_t)$, as the absolute difference of acceptance rates between the two clients, i.e., $\texttt{Imbalance}(\vec{z}_t) \coloneqq \left|\sum_{i=1}^{t} \mathbf{1}(d_i = A \wedge a_i \neq 0) - \sum_{i=1}^{t} \mathbf{1}(d_i = B \wedge b_i \neq 0)\right|$. Then, the synthesis problem for balanced servers can be encoded using Prob. 1 by assigning $\texttt{rew}(\vec{z}_t) = \sum_{i=1}^{t} \texttt{Profit}(a_i, b_i, d_i)$ and $\texttt{cost}(\vec{z}_t) = 1$ iff $\texttt{Imbalance}(\vec{z}_t) \leq \delta$ for a given tolerance $\delta > 0$.

▶ **Example 9** (Maximally responsive server)**.** The objective of the maximally responsive server is to minimize the maximum waiting time for each client while paying a bounded total amount of the sum of opportunity costs. As for the balanced server problem, we assume that the distribution θ is known and does not change over time and we have given an arbitrary trace $\vec{z}_t = (a_1, b_1)d_1 \dots (a_t, b_t)d_t \in (\mathcal{X} \times \mathcal{Y})^t$. We define the *maximum waiting time*, denoted as $\texttt{MaxWait}(d_1 \dots d_t)$, to be equal to the length of the largest subsequence of consecutive A's or B's in the sequence $d_1 \dots d_t$. Additionally, we define the *opportunity cost* at stage i, denoted as $\texttt{OppCost}(a_i, b_i, d_i)$, to be equal to $|a_i - b_i|$ if $a_i > b_i$ but $d_i = B$ or if $a_i < b_i$ but $d_i = A$. Then, the synthesis problem for maximally responsive servers can be encoded using Prob. 1 by assigning $\texttt{rew}(\vec{z}_t) = -\texttt{MaxWait}(d_1 \dots d_t)$ and $\texttt{cost}(\vec{z}_t) = 1$ iff $\sum_{i=1}^{t} \texttt{OppCost}(a_i, b_i, d_i) \leq k$ for a given budget $k > 0$, and is 0 otherwise.

▶ **Example 10** (Clientele-aware server)**.** Clientele-aware servers account for the loss of clients due to dissatisfied clients who were rejected in the past. For simplicity, we assume that only client A is active and B is inactive; this can be achieved by simply setting $b_i = 0$ always. For simplicity, we will write $(a_i, \cdot)$ to denote $(a_i, b_i = 0)$. The critical component in this example is the environment θ. We assume $\theta(\varepsilon)$, for the empty word ε, is the given initial distribution $\mathbb{P}_0$ over $\mathcal{X}$ such that the probability of seeing the price a from the client A conditioned on $a \neq 0$ is fixed as $\mathbb{P}_A$, i.e., $\mathbb{P}_0((a, \cdot) \mid a \neq 0) = \mathbb{P}_A(a)$. We assume that a constant $\delta \in (0, 1)$ is given such that for every $\vec{z}_i = (a_1, \cdot)d_1 \dots (a_i, \cdot)d_i \in (\mathcal{X} \times \mathcal{Y})^i$, we have:

$$\theta(\vec{z}_i)((a = 0, \cdot)) = \mathbb{P}_0((a_0 = 0, \cdot)) + \max\left\{0, \min\left\{1, \delta \cdot \sum_{j=1}^{i-1} \left(\mathbf{1}(a_j \neq 0 \wedge d_j = A) - \mathbf{1}(a_j \neq 0 \wedge d_j \neq A)\right)\right\}\right\},$$

and $\theta(\vec{z}_i)((a, \cdot) \mid a \neq 0) = \mathbb{P}_A(a)$ for every $a \in [1; C]$. Intuitively, the probability of seeing "$a = 0$" increases or decreases by δ for every rejection or acceptance of A, respectively. Conditioned on "$a \neq 0$" being true, the probability of seeing a price a from A is fixed to $\mathbb{P}_A(a)$. The reward and cost functions are: $\texttt{rew}(\vec{z}_t) = \sum_{i=1}^{t} \texttt{Profit}(a_i, \cdot, d_i) = \sum_{i=1}^{t} a_i \cdot \mathbf{1}(d_i = A)$ and $\texttt{cost}(\vec{z}_t) = 1$ iff $\sum_{i=1}^{t} \mathbf{1}(d_i = A) \leq N$ for a given N.

2.4 Specifications using Counter Automata

Counter automaton is a rich framework for modeling functions that has access to both states and a finite set of counters. When $\texttt{rew}$, $\texttt{cost}$, and θ are given as counter automata, we show that there is a systematic procedure to extract the statistical complexity and the witness statistic μ with theor respective μ-representations $\widehat{\texttt{rew}}$, $\widehat{\texttt{cost}}$, and $\widehat{\theta}$.

Our counter automata are adoptions of counter monitors introduced by Ferrère et al. [6]. A counter is an integer variable, which can be read and written according to relations and functions in the signature $S = \langle 0, +1, -1, \leq \rangle$. In particular, a *test* is a conjunction of atomic formulas over S and their negation, and an *update* is a mapping from variables to terms over S. The sets of tests and updates over a set of counters X are denoted as $\Phi(X)$ and $\Gamma(X)$, respectively.

▶ **Definition 11** (Counter automata). *Let Σ be a given input alphabet and D be an output domain. A* counter automaton *A on Σ and D is a tuple $\langle \Sigma, D, R, Q, \lambda, q_{\mathsf{init}}, \delta \rangle$, where R is a finite set of registers, Q is a finite set of states, $\lambda \colon Q \times \mathbb{N}^R \to D$ is an output function, $q_{\mathsf{init}} \in Q$ is the initial state, and $\delta \subseteq Q \times \Sigma \times \Phi(R) \times \Gamma(R) \times Q$ is a transition relation such that for every state $q \in Q$, input $\sigma \in \Sigma$, and valuation $v \colon R \to \mathbb{N}$, there exists a unique edge $(q, \sigma, \phi, \gamma, q') \in \delta$ with $v \models \phi$ satisfied.*

A run of the automaton on a given finite word $w = w_0 w_1 \dots w_n \in \Sigma^*$ is the unique sequence of transitions $(q_0, v_0) \xrightarrow{w_0} (q_1, v_1) \xrightarrow{w_1} \dots \xrightarrow{w_n} (q_n, v_n)$ such that $q_0 = q_{\mathsf{init}}$, $v_0(r) = 0$ for every $r \in R$, and we write $(q, v) \xrightarrow{\sigma} (q', v')$ when there exists an edge $(q, \sigma, \phi, \gamma, q') \in \delta$ such that $v \models \phi$ and $v'(r) = v(\gamma(r))$ for every $r \in R$. We write $\mathit{final}_A(w)$ to denote the last configuration (q_n, v_n). The semantics of a counter automaton A is given as $[\![A]\!](w) = \lambda(\mathit{final}_A(w)) \in D$ where (q, v) is the final configuration of the run of A on w.

▶ **Proposition 12.** *Suppose $\varphi \colon \Sigma^* \to D$ is a function and A_φ is the equivalent counter automaton on Σ such that $\varphi(w) = [\![A_\varphi]\!](w)$ for every $w \in \Sigma^*$. The function φ is μ-representible for the statistic μ defined as $\mu(w) \coloneqq \mathit{final}_{A_\varphi}(w)$, and the μ-representation of φ is given by the output function of A_φ.*

When θ, `rew`, and `cost` are expressed using counter automata, Prop. 12 outlines a simple syntactic approach to extract the well-behaved statistic μ and the respective μ-representation. The output domains of the counter-automata for θ, `rew`, and `cost` are, respectively, $\mathbb{R}$, $\{0, 1\}$, and $\Delta(\mathcal{X})$. Fig. 1 shows examples of counter automata representations of some of the statistical properties from Ex. 8–10.

For any given counter automaton $A = \langle \Sigma, D, R, Q, \lambda, q_{\mathsf{init}}, \delta \rangle$, the size of the underlying statistic μ as defined in Prop. 12 is given by:

$$\mathit{size}_\mu(t) = |\{\mathit{final}_A(w) \mid w \in \Sigma^t\}| = |Q| \times \underbrace{t \times |R|}_{\mathrm{P}}, \tag{3}$$

where P is the total number of counter values, obtained by using the fact that each counter will have a value in $[0; t-1]$ after t stages.

3 Policy Synthesis Algorithms

3.1 Dynamic Programming

Prob. 1 can be solved using dynamic programming, whose time and space complexities would unfortunately grow exponentially with respect to the horizon T. Let $\vec{z}_t$ be a given trace of length $t \in [0; T]$ and $x \in \mathcal{X}$ be an input; $\vec{z}_0$ is the empty word. We write $\mathtt{v}_t(\vec{z}_t)$ and $\mathtt{w}_t(\vec{z}_t, x)$ to denote the maximum expected rewards achievable in the remaining $T - t$ stages, while fulfilling the cost constraint, *before* and *after* revealing the next input x, respectively. We can compute the functions $\{\mathtt{v}_t(\cdot)\}_{t \in [0;T]}$ and $\{\mathtt{w}_t(\cdot, \cdot)\}_{t \in [1;T]}$ recursively as below:

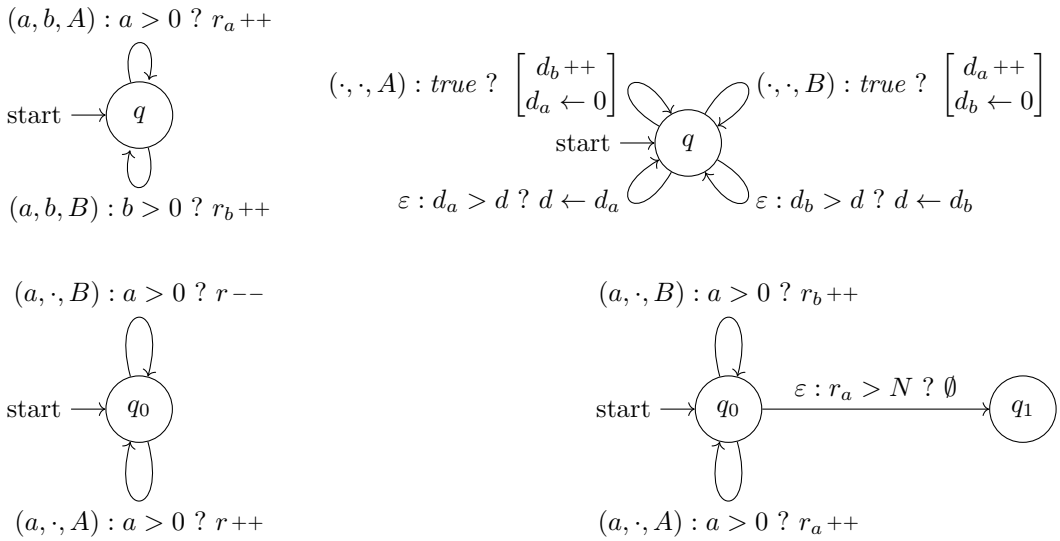


Figure 1 Counter automata representation of several properties used in Ex. 8, 9, and 10. TOP-LEFT represents `Imbalance` from Ex. 8: The counters r_a and r_b count the total numbers of accepted nonzero price requests from the clients A and B, respectively. For `Imbalance`, we use the output function $\lambda(q, r_a, r_b) = 1$ if $|r_a - r_b| \leq \delta$ and $\lambda(q, r_a, r_b) = 0$ otherwise. TOP-RIGHT represents `MaxWait` from Ex. 9: The counters d_a and d_b count the total numbers of stages since requests from, respectively, clients A and B were accepted (i.e., the current waiting times). The counter d keeps track of the current maximum waiting times for both clients. For `MaxWait`, we use the output function $\lambda(q, d_a, d_b, d) = d$. BOTTOM-LEFT represents the environment from Ex. 10: The counter r represents the difference between the numbers of acceptances and rejections of offers by A that were positive. For the environment, we use the output function $\lambda(q, r)$ that is a probability distribution over $[0; C]$ such that $\lambda(q, r)(0) = \mathbb{P}_0((a = 0)) + \max\{0, \min\{1, r\}\}$. BOTTOM-RIGHT represents the cost function from Ex. 10: The counters r_a and r_b represent the numbers of times A and B were accepted, respectively, given the price offered by A was positive. For the constraint, we use the output function $\lambda(q_0, r_a, r_b) = 1$ and $\lambda(q_1, r_a, r_b) = 0$.

$$
\begin{aligned}
t \in [0; T-1]: \quad & \mathtt{v}_t(\vec{z}_t) = \sum_{x \in \mathcal{X}} \mathtt{w}_t(\vec{z}_t, x) \cdot \theta(\vec{z}_t)(x), \\
& \mathtt{w}_t(\vec{z}_t, x) = \max_{y \in \mathcal{Y}} \mathtt{v}_{t+1}(\vec{z}_t x y), \\
t = T: \quad & \mathtt{v}_t(\vec{z}_{t=T}) = \begin{cases} -\infty & \text{if } \mathtt{cost}(\vec{z}_T) = 0, \\ \mathtt{rew}(\vec{z}_T) & \text{otherwise.} \end{cases}
\end{aligned}
\tag{\texttt{standard-DP}}
$$

The sought optimal transducer γ^* of Prob. 1 is then obtained as: for every $t \in [0; T-1]$ and for every $\vec{z}_t \in (\mathcal{X} \times \mathcal{Y})^t$, if $\mathtt{v}_t(\vec{z}_t) \neq -\infty$ then $\gamma(\vec{z}_t x) = \arg\max_{y \in \mathcal{Y}} \mathtt{v}_{t+1}(\vec{z}_t x y)$, and else $\gamma(\vec{z}_t x) = \mathtt{FAIL}$. Note that even if $\mathtt{v}_0(\varepsilon) \neq -\infty$, for the empty word ε, the environmental uncertainty may force the system to a stage from where the cost constraint can no longer be satisfied, which is when γ would output `FAIL`.

As we unroll the recursion tree in (`standard-DP`) forward, we observe that every $\mathtt{v}_t$ node has $|\mathcal{X}|$ children and every $\mathtt{w}_t$ node has $|\mathcal{Y}|$ children. Therefore, the size of the entire tree will be $\mathcal{O}(|\mathcal{X}|^T \cdot |\mathcal{Y}|^T)$, causing an exponential blow-up in time and space complexities. In fact, from a classical result by Papadimitriou [12], it follows that Prob. 1 is PSPACE-hard. Note that the paper considers the setting without the cost function `cost`, which is a special case of our setting. The paper has proven this special case to be PSPACE-complete.

The difficulty of Prob. 1 stems from the *history dependence* of the stochastic generator θ, the reward function $\texttt{rew}$, and the cost function $\texttt{cost}$. Take for example the case where the output symbol selected at a given stage affects the distribution of θ at a future stage, for which building the entire recursion tree is unavoidable [12]. A similar situation arises when the reward and the cost are affected by all the past inputs and outputs and their exact order.

3.2 Simple Problem Instances

We take a slight detour and present a subclass of problem instances for which a PTIME algorithm exists; ideas from this algorithm will be useful in Sec. 3.4 when we will present special optimizations of our algorithm. We call the problem instance $\langle \mathcal{X}, \mathcal{Y}, \theta, \texttt{rew}, \texttt{cost}, T\rangle$ *simple* if the following hold:

Additive reward and cost: For every $t \in [1;T]$, let there be functions $r_t, c_t \colon \mathcal{X} \times \mathcal{Y} \to \mathbb{R}_{\geq 0}$, assigning constant, history-independent reward and cost to each individual input-output pair, such that for every $\vec{z} = x_1y_1x_2y_2\ldots$, $\texttt{rew}(\vec{z}) = \sum_t r_t(x_ty_t)$ and $\texttt{cost}(\vec{z}) = 1$ iff $\sum_t c_t(x_ty_t) \leq B$ for a given *budget* B.

History-independent environment: For every $t \in [0;T-1]$, let $\bar{\theta}_t$ be a fixed probability distribution such that $\theta(\vec{w}_t) = \theta(\vec{z}_t) = \bar{\theta}_t$ for every $\vec{w}_t, \vec{z}_t \in (\mathcal{X}\times\mathcal{Y})^t$.

Then we can modify the dynamic programming algorithm in (`standard-DP`) by replacing the history $\vec{z}$ with just the cost b incurred in $\vec{z}$, and define the respective counterparts $\texttt{v}'_t(b), \texttt{w}'_t(b,x)$ of $\texttt{v}_t(\vec{z}), \texttt{w}_t(\vec{z},x)$ as below:

$$
\begin{aligned}
t \in [0:T-1]: \quad & \texttt{v}'_t(b) = \sum_{x\in\mathcal{X}} \texttt{w}'_t(b,x)\cdot\bar{\theta}_t(x), \\
& \texttt{w}'_t(b,x) = \max_{y\in\mathcal{Y}} \left[\texttt{v}'_{t+1}(b + c_t(x,y)) + r_t(x,y)\right], \\
t = T: \quad & \texttt{v}'_t(b) = \begin{cases} -\infty & \text{if } b > B, \\ 0 & \text{otherwise.} \end{cases}
\end{aligned}
\tag{\texttt{simple-DP}}
$$

The sought optimal transducer γ^* of Prob. 1 is obtained as follows: for every $t \in [0;T-1]$ and for every $\vec{z}_t \in (\mathcal{X}\times\mathcal{Y})^t$ with $b = \sum_{i=1}^t c_i(x_iy_i)$, if $\texttt{v}'_t(b) \neq -\infty$ then $\gamma(\vec{z}_tx) = \arg\max_{y\in\mathcal{Y}} \texttt{v}'_{t+1}(b + c_t(xy))$, and else $\gamma(\vec{z}_tx) = \texttt{FAIL}$, where `FAIL` is defined as for the case of `standard-DP`, and indicates that the cost constraint cannot be satisfied in any possible future extension of the current trace.

Let us analyze the size of the recursion tree of (`simple-DP`). At every stage t, the number of $\texttt{v}'_t(b)$ nodes is B and the number of $\texttt{w}'_t(b,x)$ nodes is $B\cdot|\mathcal{X}|$, implying that the tree's size is $\mathcal{O}(T\cdot B\cdot|\mathcal{X}|)$, establishing the PTIME complexity.

In general, if even one of the three functions $\texttt{rew}, \texttt{cost}$, and θ is history-dependent, like in the examples of Sec. 2.3, the PTIME algorithm is not applicable. In the next section, we show a sub-class of Prob. 1 with statistical properties – with the dependence being only on a small statistic over the history instead of the exact history – for which efficient algorithms exist.

3.3 Statistical Abstraction

We now present the main contribution of this paper. Suppose we are given a problem instance $\langle \mathcal{X}, \mathcal{Y}, \theta, \texttt{rew}, \texttt{cost}, T\rangle$ such that there exists a well-behaved $\mathcal{S}$-statistic μ on $(\mathcal{X}\times\mathcal{Y})^*$ for which θ, $\texttt{rew}$, and $\texttt{cost}$ are μ-representible. In this case, we no longer need to consider each history $\vec{z}_t$ while using `standard-DP`, but rather we can combine histories that are statistically indistinguishable, i.e., equivalent with respect to $\equiv_{\mu,t}$ for a given t. This way

we obtain an abstraction-based implementation of `standard-DP`– called `statistical-DP`, where, for each stage t, $(\mathcal{X} \times \mathcal{Y})^t$ serves as the concrete domain of histories, $\mathcal{S}$ serves as its abstraction, μ serves as the abstraction function, and κ_t serves as the concretization function. Following we give a key consistency property of the abstraction, whose proof follows from the well-behavedness of μ.

▶ **Proposition 13.** *For a fixed stage $t < T$ and an arbitrary history $\vec{z}_t \in (\mathcal{X} \times \mathcal{Y})^t$ with $\mu(\vec{z}_t) = s$, it holds that for every $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, we have $\mu(\vec{z}_t xy) = \mu(\kappa_t(s)xy)$.*

The above claim suggests that the statistic μ of an input-output trace can be computed sequentially and efficiently, without keeping track of the entire history but only from the value of the statistic in the last step. This enables us to only keep track of the value of the statistic μ in each recursive call, instead of the full histories and lets us lift the functions $\mathtt{v}_t$ and $\mathtt{w}_t$ to their abstract counterparts, respectively, $\widehat{\mathtt{v}}_t$ and $\widehat{\mathtt{w}}_t$ as follows. For every stage $t \in [0;T]$, we define $\widehat{\mathtt{v}}_t \colon \mathcal{S} \to \mathbb{R}$ and $\widehat{\mathtt{w}} \colon \mathcal{S} \to \{0,1\}$ as below:

$$
\begin{aligned}
t \in [0;T-1]: \quad \widehat{\mathtt{v}}_t(s) &= \sum_{x \in \mathcal{X}} \widehat{\mathtt{w}}_t(s,x) \cdot \overbrace{\widehat{\theta}(s)(x)}^{\textbf{I}}, \\
\widehat{\mathtt{w}}_t(s,x) &= \max_{y \in \mathcal{Y}} \overbrace{\widehat{\mathtt{v}}_{t+1}\left(\mu(\kappa_t(s)xy)\right)}^{\textbf{II}}, \qquad \text{(statistical-DP)} \\
t = T: \quad \widehat{\mathtt{v}}_t(s) &= \begin{cases} -\infty & \text{if } \widehat{\mathtt{cost}}(s) = 0 \quad \leftarrow \textbf{IV} \\ \underbrace{\widehat{\mathtt{rew}}(s)}_{\textbf{III}} & \text{otherwise.} \end{cases}
\end{aligned}
$$

We use the annotations **I**, **II**, **III**, and **IV** in Sec. 3.4 to present further optimizations.

The sought optimal transducer γ^* of Prob. 1 is obtained as follows: for every $t \in [0;T-1]$ and every $\vec{z}_t \in (\mathcal{X} \times \mathcal{Y})^t$, if $\widehat{\mathtt{v}}_t(\mu(\vec{z}_t)) \neq -\infty$ then $\gamma(\vec{z}_t x) = \arg\max_{y \in \mathcal{Y}} \widehat{\mathtt{v}}_{t+1}(\mu(\vec{z}_t xy))$, and else $\gamma(\vec{z}_t x) = \mathtt{FAIL}$, where `FAIL` is as defined for the case of `standard-DP`, and indicates that the cost constraint cannot be satisfied in any possible future extension of the current trace.

Next we compare `statistical-DP` from above with the general dynamic programming scheme `standard-DP` for Prob. 1. The highlight of `statistical-DP` is that the functions $\widehat{\mathtt{v}}_t$ and $\widehat{\mathtt{w}}_t$ do not depend on the history $\vec{z}_t$ (unlike $\mathtt{v}_t$ and $\mathtt{w}_t$ in `standard-DP`) anymore; rather they only depend on the statistical values of histories. If we interpret each $\widehat{\mathtt{v}}_t(\cdot)$ and $\widehat{\mathtt{w}}_t(\cdot, x)$ as vectors V_t and W_t, respectively, then the sizes of V_t and W_t are equal to the size $size_\mu(t)$ of the statistic μ on t, and not on the size of $(\mathcal{X} \times \mathcal{Y})^t$. Usually, $size_\mu(t)$ is substantially smaller than $|(\mathcal{X} \times \mathcal{Y})^t|$, and the computational savings with `statistical-DP` are significant. For instance, in Ex. 8 and Ex. 9, when the property is specified using the counter automata in Fig. 1, it follows from (3) that $size_\mu(t) = 1 \times t \times 4 = 4t$, which is significantly smaller than $|(\mathcal{X} \times \mathcal{Y})^t| = (|G| \times |C| \times |E|)^t = (2 \times 2c \times 2)^t = (8c)^t$ for sufficiently large t.

Theorem 14 demonstrates the correctness and the complexity of `statistical-DP`.

▶ **Theorem 14.** *Let $\langle \mathcal{X}, \mathcal{Y}, \theta, \mathtt{rew}, \mathtt{cost}, T \rangle$ be a problem instance and μ be a well-behaved statistic on $(\mathcal{X} \times \mathcal{Y})^*$ such that θ, `rew`, and `cost` are μ-representible. For every stage $t \in [0;T]$, and every $\vec{z}_t \in (\mathcal{X} \times \mathcal{Y})^t$ with $\mu(\vec{z}_t) = s$ we have $\mathtt{v}_t(\vec{z}_t) = \widehat{\mathtt{v}}_t(s)$. Furthermore, the computation of $\{\widehat{\mathtt{v}}_t(\cdot)\}_{t \in [0;T]}$ requires $\mathcal{O}\left(|\mathcal{X}| \cdot |\mathcal{Y}| \cdot \sum_{t=1}^{T} size_\mu(t)\right)$ time and $\mathcal{O}\left(\sum_{t=1}^{T} size_\mu(t)\right)$ space.*

▶ Remark 15 (Product statistic). If θ, `rew`, and `cost` are represented using individual well-behaved statistics μ_θ, $\mu_{\mathtt{rew}}$, and $\mu_{\mathtt{cost}}$, respectively, the common statistic μ of Thm. 14 can be obtained by computing a product statistic as follows: For every $\vec{z}_t$, $\mu(\vec{z}_t) :=$

$(\mu_\theta(\vec{z}_t), \mu_{\texttt{rew}}(\vec{z}_t), \mu_{\texttt{cost}}(\vec{z}_t)$. If θ, `rew`, and `cost` are provided as counter automata, then μ can be extracted from their product automaton. Consequently, we obtain $size_\mu(t) = size_{\mu_\theta}(t) \times size_{\mu_{\texttt{rew}}}(t) \times size_{\mu_{\texttt{cost}}}(t)$.

Proof of Thm. 14. The first part of the proof is via backward induction on the stages. Throughout, we will use the trace $\vec{z}_t \in (\mathcal{X} \times \mathcal{Y})^t$ for which $\mu(\vec{z}_t) = s$. To make the proof easy to follow, we show that both (i) $\mathtt{v}_t(\vec{z}_t) = \widehat{\mathtt{v}}_t(s)$ and (ii) $\mathtt{w}_t(\vec{z}_t, x) = \widehat{\mathtt{w}}_t(s, x)$ hold.

For the base case $t = T$, from Prop. 6 and the μ-representability of `rew` and `cost`, it follows that $\texttt{rew}(\vec{z}_t) = \texttt{rew}(\kappa_t \circ \mu(\vec{z}_t)) = \widehat{\texttt{rew}}(s)$ and $\texttt{cost}(\vec{z}_t) = \texttt{cost}(\kappa_t \circ \mu(\vec{z}_t)) = \widehat{\texttt{cost}}(s)$. Therefore, $\mathtt{v}_T(\vec{z}_T) = \widehat{\mathtt{v}}_T(s)$ holds as the base case for Claim (i). From the base case and Prop. 13 we obtain that, for every $t \in [0; T-1]$, it holds that:

$$\begin{aligned}\mathtt{w}_t(\vec{z}_t, x) &= \max_{y \in \mathcal{Y}} \mathtt{v}_{t+1}(\vec{z}_t xy) = \max_{y \in \mathcal{Y}} \widehat{\mathtt{v}}_{t+1}(\mu(\vec{z}_t xy)) \\ &= \max_{y \in \mathcal{Y}} \widehat{\mathtt{v}}_{t+1}(\mu(\kappa_t(s) xy)) = \widehat{\mathtt{w}}_t(s, x). \end{aligned} \tag{4}$$

By using $t = T-1$, we obtain $\mathtt{w}_{T-1}(\vec{z}_{T-1}, x) = \widehat{\mathtt{w}}_{T-1}(\mu(\vec{z}_{T-1}), x)$, which serves as the base case for Claim (ii).

Now suppose Claim (i) holds for an arbitrary stage $t+1 \leq T$. Then the same derivation (4) can be used to show that Claim (ii) holds for the stage t. These two statements serve as the induction steps, and we will show that Claim (i) holds for stage t and Claim (ii) holds for stage $t-1$. The following is the proof of the induction step for Claim (i) which holds due to the base case and the μ-representibility of θ:

$$\mathtt{v}_t(\vec{z}_t) = \sum_{x \in \mathcal{X}} \mathtt{w}_t(\vec{z}_t, x) \cdot \theta(\vec{z}_t)(x) = \sum_{x \in \mathcal{X}} \widehat{\mathtt{w}}_t(s, x) \cdot \widehat{\theta}(s)(x) = \widehat{\mathtt{v}}_t(s). \tag{5}$$

We can use the same derivation (4) by only substituting $t+1$ with t, where the second inequality will now follow from (5), and obtain the proof of induction step for Claim (ii).

The time and space complexity bounds are established as follows. It follows from (3) that for each stage t, there are at most $size_\mu(t)$ distinct values of $s \in \mathcal{S}$ which can represent the whole set $(\mathcal{X} \times \mathcal{Y})^t$. Then, $\widehat{\mathtt{v}}_t(\cdot)$ will have at most $size_\mu(t)$ arguments. Therefore, there are $\mathcal{O}(\sum_{t=1}^T size_\mu(t))$ elements in $\{\widehat{\mathtt{v}}_t(\cdot)\}_{t \in [1;T]}$, for which we will need $\mathcal{O}(\sum_{t=1}^T size_\mu(t))$ space in total. On the other hand, for every distinct element in $\{\widehat{\mathtt{v}}_t\}_{t \in [1;T]}$, the recursion will have $|\mathcal{X}| \times |\mathcal{Y}|$ branchings ($|\mathcal{X}|$ branchings for the sum operator and $|\mathcal{Y}|$ branchings for the max). Therefore, we will need $\mathcal{O}(|\mathcal{X}| \cdot |\mathcal{Y}| \cdot \sum_{t=1}^T size_\mu(t))$ time in total for the computation. ◀

3.4 Additional Optimizations for Special Cases

We identify three special cases with potentially additional complexity improvements. We highlight conditions under which we can disregard each individual size, $size_{\mu_\theta}$, $size_{\mu_{\texttt{rew}}}$, and $size_{\mu_{\texttt{cost}}}$, one by one, when computing $size_\mu$, as outlined in Remark 15. This optimization potentially reduces $size_\mu$, providing synthesis algorithms with improved complexity.

Independent environmental distributions. Suppose θ has the property that for every $t \in [0; T-1]$ and for every $\vec{w}_t, \vec{z}_t \in (\mathcal{X} \times \mathcal{Y})^t$, we have $\theta(\vec{w}_t) \equiv \theta(\vec{z}_t)$ (in distribution), which we write as θ_t. We can replace $\widehat{\theta}(s)(x)$ in (**I**) in `statistical-DP` with $\theta_t(x)$. Removing $\widehat{\theta}$ from `statistical-DP` means that we can ignore $size_{\mu_\theta}$ while computing $size_\mu$ of the product statistic μ as in Rem. 15 (i.e., set $size_{\mu_\theta}(t) = 1$ for all t).

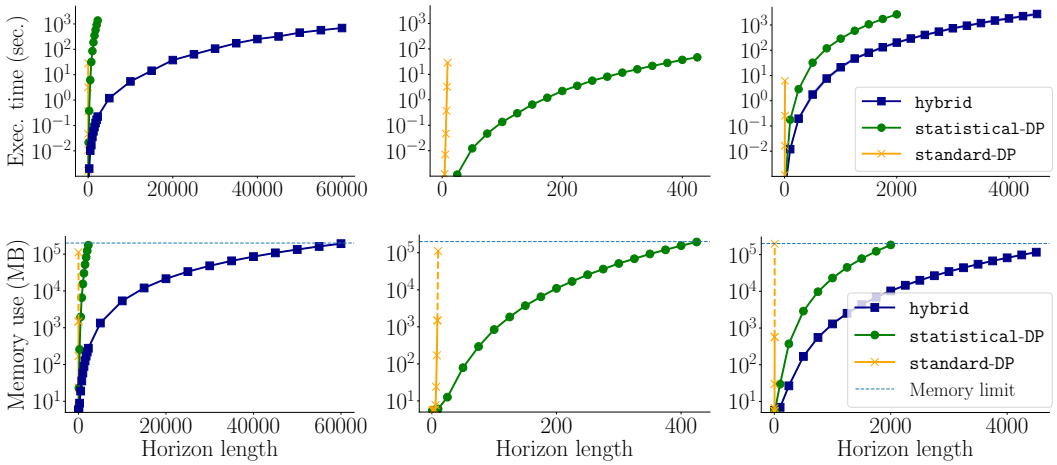


Figure 2 Top row: plots of computation time (in log-scale) versus horizon length for Ex. 8, 9, and 10 (left to right). Bottom row: plots of memory usage (in log-scale) versus horizon length for Ex. 8, 9, and 10 (left to right).

Additive reward. Suppose `rew` has the property that for every $\vec{z}_t = x_1y_1 \dots x_ty_t \in (\mathcal{X} \times \mathcal{Y})^t$, we can express the reward as $\texttt{rew}(\vec{z}_t) = \sum_{i=1}^{t} r(x_1y_1)$ for some positive, stage-invariant reward $r\colon \mathcal{X} \times \mathcal{Y} \to \mathbb{R}_{\geq 0}$. In `statistical-DP`, we can replace $\widehat{\mathrm{v}}_{t+1}(\mu(\kappa_t(s)xy))$ in (**II**) with $\widehat{\mathrm{v}}_{t+1}(\mu(\kappa_t(s)xy)) + r(xy)$ and replace $\widehat{\texttt{rew}}(s)$ in (**III**) with "0," like in the simple case described in Sec. 3.2. Removing $\widehat{\texttt{rew}}$ from `statistical-DP` means that we can ignore $size_{\mu_{\texttt{rew}}}$ while computing $size_\mu$ of the product statistic μ as in Rem. 15. Both Ex. 8 and Ex. 10 have additive rewards, which helped us to obtain faster computations in our experiments; see Sec. 4.1.

Cost is a safety constraint. Suppose `cost` is a stateless, boolean safety property, specified using a function $c\colon \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$ such that $\texttt{cost}(x_1y_1 \dots x_ty_t) = 0$ iff there exists an $i \in [1; t]$ for which $c(x_iy_i) = 1$. In `statistical-DP`, we can replace (**II**) with "0" if $c(xy) = 1$, and remove (**IV**). Removing $\widehat{\texttt{cost}}$ from `statistical-DP` means that we can ignore $size_{\mu_{\texttt{cost}}}$ while computing $size_\mu$ of the product statistic μ as in Rem. 15.

4 Experiments

4.1 Comparison of Computational Performances

We considered problem instances with varying time horizons for all three of our examples, namely synthesis of balanced, maximally responsive, and clientele-aware servers, formalized in Ex. 8, 9, and 10, respectively. For each example, we compared the computational performances between `standard-DP` and `statistical-DP`, and for Ex. 8 and Ex. 10, we also compared the hybrid approach discussed in Sec. 3.4 for when the reward is additive. Fig. 2 compares the time and memory usage of our synthesis tool to solve each problem instance using different approaches. For all reported results we use a time limit of 1 hour and a memory limit of 200 GB. As expected, the time and memory usage for `standard-DP` grows exponentially with T in all cases, and the memory limit is reached way earlier compared to the `statistical-DP` approach. The hybrid approach, when available, performs even better than `statistical-DP`.

4.2 Analysis of Synthesized Policies

We demonstrate the quality of the policies synthesized using `statistical-DP` for each of the three examples that we consider. Tab. 1 summarizes the problem instances and Fig. 3 summarizes the aggregated outcomes of 100 simulations in each case.

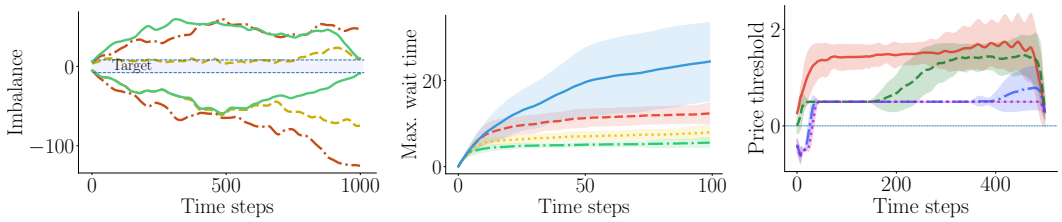
Example – Balanced server. We compare the quality of the synthesized balanced server policy, formalized in Ex. 8 and from hereon referred to as the *balanced-by-construction* policy, with two baseline policies. The first baseline is a *greedy* policy, that always accepts the client with the highest price, and in the case of a tie, accepts the client that would make the imbalance smaller. The second baseline maximizes the expected profit, constrained to having a balanced execution in expectation. We call this the *balanced-on-average* policy. Fig. 3a shows that the *balanced-by-construction policy is balanced on every run*, whereas both the greedy policy and the balanced-on-average policy are unbalanced in the worst case. Besides, the average profit of the balanced-by-construction policy remains competitive (not shown in figures). In particular, on average the profit for the balanced-by-construction policy, the greedy policy, and the balanced-on-average policy were, respectively: 10719, 10723, and 8267. In this setting, the balanced-by-construction policy obtains almost the same profit as the greedy policy, while maintaining the balance constraint. The balanced-on-average policy obtains worse results both in terms of profit and balance.

Example – Maximally responsive server. We compare the performance of a maximally responsive server, formalized in Ex. 9, with the same input distribution and different budgets. As a baseline, we use a greedy policy that accepts the client offering the highest price. Fig. 3b shows the average and standard deviation of the maximal response time achieved by policies with different bounds on the total opportunity costs. In the experiments illustrated in Fig. 3b, the greedy policy obtains an average maximum response time of 24.4 time steps after a horizon length of 100 time steps. Our maximally responsive server with budgets 5, 15, and 25, respectively, achieves average maximum waiting time of 12.35, 7.94, and 5.55 while obtaining 98.75%, 96.25%, and 93.81% of the profit obtained by the greedy policy. Intuitively, the higher the budget on opportunity costs is, the more freedom the server gets to reduce the waiting time, though this impacts the overall profit.

Example – Clientele-aware server. We compare the performances of four different clientele-aware server policies (from Ex. 10) with four different constraints on the maximum number N of accepted requests. From the plot in Fig. 3c, we observe that the higher the value of N is, the lower is the initial price threshold, i.e., the higher is the acceptance rate in the beginning. This happens because for a higher value of N, the server has more freedom to initially accept more requests from the client (by lowering the threshold) to improve the chances of seeing better offer prices in future.

Table 1 Where $q_{n,k}$ is the centered $\text{Binomial}_{n,k/n}$, i.e. $X - \mathbb{E}(X) \sim q_{n,k}$ with $X \sim \text{Binomial}_{n,k/n}$.

Example	Horizon	Price A PDF	Price B PDF	Objective	Constraint
8	1000	$\text{Binomial}_{20,0.49}$	$\text{Binomial}_{20,0.51}$	profit	imbalance ($\delta = 5$)
9	100	$\text{Binomial}_{6,0.3}$	$\text{Binomial}_{6,0.75}$	wait time	budget ($k \in \{5, 15, 25\}$)
10	500	$q_{6,1}$	-	profit	budget ($N \in \{50, 150, 250, 350\}$)



(a) Balanced server. **(b)** Maximally responsive. **(c)** Clientele-aware.

Figure 3 **Fig. 3a** depicts the minimal and maximal imbalance values over 100 simulations for the balanced-by-construction policy (——), the greedy policy (– – –), and the balanced-on-average policy (– – –). **Fig. 3b** depicts the maximal wait time values obtained over 100 simulations after deploying the maximally responsive server for a budget of 5 (– –), 10 (····), and 25 (–·–·–). We also show for comparison, the maximal response time obtained by a server with a greedy policy (——). **Fig. 3c** depicts mean and standard deviation of the acceptance thresholds of the synthesized policy for various budget constraints: 50 (——), 150 (– – –), 250 (– – –), and 350 (····).

5 Related Work

The literature on bounded-horizon sequential decision-making broadly distinguishes between *model-free* and *model-based* instances of the problem, based on whether a model of the environment is available [14]. In this paper, we study the model-based problem.

For model-based problems, a common assumption is that the environment can be modeled with a Markov Decision Process (MDP) [14]. Hence, the synthesis problem reduces to finding the optimal policy in a finite MDP [13], for which extensive literature is available [7]. The size of the MDP can however be exponential in the horizon length, and solving the derived optimization problem is known to be PSPACE-hard [12]. Our statistical abstraction can be viewed as a small abstraction over the state space of this exponentially large MDP, which is only implicitly built and explored via `statistical-DP`. A large body of work on model-based sequential decision-making falls under the term *optimal stopping problems* [15]. The theory of optimal stopping revolves around the problem of choosing the ideal time to take a specific action to either maximize an expected reward or minimize an expected cost.Among those works, a variety of environment models – both Markovian [15] and non-Markovian [3] – and a variety of properties [1, 4, 11, 8] have been studied. To the best of our knowledge, the history-dependent statistical properties that we consider are beyond the reach of the existing algorithms in the optimal stopping literature. Moreover, due to the generality of our assumptions, common analytical tools, like competitive ratios [16], would fail to provide anything beyond trivial bounds.

For model-free problems, optimal policy synthesis algorithms are predominantly data-driven and rely on learning [18]. Under the assumption that the unknown state space is finite, PAC-style guarantees are possible [17]. The quality of algorithms in this area is often assessed using regret where the performance is compared against the model-based setting [5, 19]. We plan to consider this direction for future work, with an unknown environment and the goal to compute policies that fulfill the objectives with high probabilities.

6 Conclusion

We considered the sequential decision-making problem with uncertain environments where the objective of the decision-maker includes fulfillment of a statistical property over a finite horizon of a given length. Although the problem was known to be PSPACE-hard for general

properties (possibly non-stochastic), for statistical properties, we present a solution whose complexity grows only polynomially with respect to the horizon length. The crux of our approach is a novel statistical abstraction that clusters statistically indistinguishable traces of the system. Using a prototype implementation, we demonstrated the computational performance and effectiveness of our approach on three examples of designing server policies that need to act fairly towards its clients.

Several future directions exist. Firstly, the unbounded-horizon setting presents an interesting and non-trivial extension, given the inherent challenge of defining the feasible set of policies (Eq. 1) within this setting. Secondly, it will be valuable to consider the multi-objective problem where both `cost` and `rew` need to be optimized. The goal here will be to compute a Pareto-optimal solution. In contrast, our approach is currently limited to unconditionally fulfilling the qualitative constraint `cost`. Finally, our work establishes only the *sufficient* requirements on the properties (μ-representibility for a well-behaved statistic μ) which admit efficient abstractions. Determining the *necessary* requirements will be an interesting theoretical quest, which may improve our understanding of the complexity landscape of sequential decision-making problems.

References

1 Stefan Ankirchner, Maike Klein, and Thomas Kruse. A verification theorem for optimal stopping problems with expectation constraints. *Applied Mathematics & Optimization*, 79:145–177, 2019.

2 Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47:235–256, 2002.

3 Elena Bandini, Andrea Cosso, Marco Fuhrman, and Huyên Pham. Backward SDEs for optimal control of partially observed path-dependent stochastic systems: a control randomization approach. *The Annals of Applied Probability*, 28(3):1634–1678, 2018.

4 Erhan Bayraktar and Song Yao. Optimal stopping with expectation constraints. *The Annals of Applied Probability*, 34(1B):917–959, 2024.

5 Yonathan Efroni, Shie Mannor, and Matteo Pirotta. Exploration-exploitation in constrained MDPs. *arXiv preprint*, 2020. `arXiv:2003.02189`.

6 Thomas Ferrère, Thomas A Henzinger, and Bernhard Kragl. Monitoring event frequencies. In *28th EACSL Annual Conference on Computer Science Logic (CSL)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

7 Abhijit Gosavi et al. *Simulation-based optimization*. Springer, 2015.

8 Sigrid Källblad. A dynamic programming approach to distribution-constrained optimal stopping. *The Annals of Applied Probability*, 32(3):1902–1928, 2022.

9 Robert Kleinberg and S Matthew Weinberg. Matroid prophet inequalities and applications to multi-dimensional mechanism design. *Games and Economic Behavior*, 113:97–115, 2019.

10 S. Muthukrishnan. Ad exchanges: Research issues. In *Internet and Network Economics, 5th International Workshop (WINE)*, pages 1–12, 2009.

11 Aaron Zeff Palmer and Alexander Vladimirsky. Optimal stopping with a probabilistic constraint. *Journal of Optimization Theory and Applications*, 175:795–817, 2017.

12 Christos H. Papadimitriou. Games against nature. *Journal of Computer and System Sciences*, 31(2):288–301, 1985.

13 David C Parkes and Satinder Singh. An MDP-based approach to online mechanism design. *Advances in neural information processing systems (NIPS)*, 16, 2003.

14 Diederik M Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 48:67–113, 2013.

15 Albert N Shiryaev. *Optimal stopping rules*, volume 8. Springer Science & Business Media, 2007.

16 Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

17 Alexander L Strehl, Lihong Li, Eric Wiewiora, John Langford, and Michael L Littman. PAC model-free reinforcement learning. In *International Conference on Machine Mearning (ICML)*, pages 881–888, 2006.

18 Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

19 Andrew J Wagenmaker, Yifang Chen, Max Simchowitz, Simon Du, and Kevin Jamieson. First-order regret in reinforcement learning with linear function approximation: A robust estimation approach. In *International Conference on Machine Learning (ICML)*, pages 22384–22429, 2022.

Lean: Past, Present, and Future

Sebastian Ullrich
Lean Focused Research Organization, USA

Abstract

The Lean programming language and theorem prover project is celebrating its tenth birthday this year, having been started by Leonardo de Moura at Microsoft Research and first release as Lean 0.1 in 2014. In this invited talk, I will review Lean's history and unique features and discuss our roadmap for its bright future.

Corresponding to its major versions ranging from Lean 0.1 to the current version of Lean 4, the focus of the Lean project has evolved over the years. Initially intended as a platform for developing *white-box* automation, in contrast to the usual black-box approach of stand-alone SMT solvers [2], the system gathered more conventional features of dependently-typed interactive theorem provers as well as an initial crowd of interested mathematicians and computer scientists with its first official release as Lean 2 in 2015 [1]. Lean 3 in 2017 introduced *user-extensible* automation by extending Lean from a specification language to an accessible metaprogramming language [4], further accelerating growth of its mathematical library that was spun out into the separate *Mathlib* project [5]. Spurred by the success but also limitations of this extensibility, we started work on the next version Lean 4 in 2018 [3] with the goal of turning Lean into a *general-purpose* programming language that would allow us to reimplement Lean in Lean itself and thereby make many more aspects of the system user-extensible, in a more efficient manner [6]. This to date largest rework of Lean's implementation was completed in 2023 with the official release of Lean 4.0.0, further supporting Mathlib's growth to more than 1.5 million lines of code at the time of writing as well as improving support for many other applications such as software verification.

In 2023, Lean also saw its largest organizational change when Leo and I created the Lean Focused Research Organization (FRO)[1] to bundle and support development of Lean in a dedicated organization for the first time. Thanks to gracious support from philanthropic sponsors, an unprecedented number of currently twelve people now work on the evolution of Lean at the Lean FRO. And there is much left to do: with our new team size, we can now support development on much more than only core features, such as documentation, a robust standard library, and user interfaces and experience as well as a return to the original topic of advanced proof automation. The Lean FRO is committed to ensuring and extending Lean's applicability in education, research, and industry and to leading it into the next decade of Lean development and beyond.

2012 ACM Subject Classification Software and its engineering → Functional languages; Security and privacy → Logic and verification

Keywords and phrases Lean, interactive theorem proving, focused research organization, history

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.3

Category Invited Talk

Funding The Lean Focused Research Organization receives philanthropic support from the Simons Foundation International, the Alfred P. Sloan Foundation, and Richard Merkin.

References

1 Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The Lean theorem prover (system description). In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction – CADE-25 – 25th International Conference on Automated Deduction, Berlin, Germany, August 1–7, 2015, Proceedings*, volume 9195 of *Lecture Notes in Computer Science*, pages 378–388. Springer, 2015. doi:10.1007/978-3-319-21401-6_26.

[1] https://lean-fro.org

2 Leonardo de Moura and Grant Olney Passmore. The strategy challenge in SMT solving. In Maria Paola Bonacina and Mark E. Stickel, editors, *Automated Reasoning and Mathematics – Essays in Memory of William W. McCune*, volume 7788 of *Lecture Notes in Computer Science*, pages 15–44. Springer, 2013. doi:10.1007/978-3-642-36675-8_2.

3 Leonardo de Moura and Sebastian Ullrich. The Lean 4 theorem prover and programming language. In André Platzer and Geoff Sutcliffe, editors, *Automated Deduction – CADE 28 – 28th International Conference on Automated Deduction, Virtual Event, July 12–15, 2021, Proceedings*, volume 12699 of *Lecture Notes in Computer Science*, pages 625–635. Springer, Springer, 2021. doi:10.1007/978-3-030-79876-5_37.

4 Gabriel Ebner, Sebastian Ullrich, Jared Roesch, Jeremy Avigad, and Leonardo de Moura. A metaprogramming framework for formal verification. *Proc. ACM Program. Lang.*, 1(ICFP):34:1–34:29, 2017. doi:10.1145/3110278.

5 The mathlib Community. The Lean mathematical library. In Jasmin Blanchette and Catalin Hritcu, editors, *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20–21, 2020*, CPP 2020, pages 367–381, New York, NY, USA, 2020. ACM. doi:10.1145/3372885.3373824.

6 Sebastian Ullrich. *An Extensible Theorem Proving Frontend*. PhD thesis, Karlsruhe Institute of Technology, Germany, 2023. doi:10.5445/IR/1000161074.

Univalent Enriched Categories and the Enriched Rezk Completion

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

Abstract

Enriched categories are categories whose sets of morphisms are enriched with extra structure. Such categories play a prominent role in the study of higher categories, homotopy theory, and the semantics of programming languages. In this paper, we study univalent enriched categories. We prove that all essentially surjective and fully faithful functors between univalent enriched categories are equivalences, and we show that every enriched category admits a Rezk completion. Finally, we use the Rezk completion for enriched categories to construct univalent enriched Kleisli categories.

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

Keywords and phrases enriched categories, univalent categories, homotopy type theory, univalent foundations, Rezk completion

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.4

Related Version *Previous Version*: `https://arxiv.org/pdf/2401.11752`

Supplementary Material *Software*: `https://doi.org/10.5281/zenodo.10849216` [33]

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

Acknowledgements The author thanks the anonymous reviewers of HoTT/UF and FSCD for their useful comments, Nima Rasekh for useful discussions, and Dan Frumin and Kobe Wullaert for proof reading earlier versions of this paper. The author also thanks the Coq developers for providing the Coq proof assistant and their continuous support to keep `UniMath` compatible with Coq.

1 Introduction

Over the years, category theory [16] has established itself as a powerful mathematical framework with a wide variety of applications. The applications of category theory range from pure mathematics [9, 35] to computer science [21, 23, 24, 25]. This study resulted in the development of various notions of categories.

One of these notions is given by *enriched categories*. Enriched categories are categories whose morphisms are equipped with additional structure. Examples of such categories are plentiful. For instance, in the study of the semantics of effectful programming languages, one uses categories enriched over *directed complete partial orders* (DCPOs) [23, 24, 25]. The type of morphisms in categories enriched over DCPOs is given by a DCPO, and thus fixpoint equations of morphisms can be solved in such categories [34]. For similar purposes, categories enriched over partial orders have been used [19]. Other applications of enriched categories appear in homological algebra [35] where one is interested in categories enriched over abelian groups, abstract homotopy theory [9] where one looks at categories enriched over simplicial sets, and higher category theory [14] where one consider categories enriched over categories.

Univalent Foundations. Throughout this paper, we work in univalent foundations [27, 31]. Univalent foundations is an extension of dependent Martin-Löf Type Theory [17] with the *univalence axiom*. This axiom says that the identity of types is the same as equivalences between them. More specifically, we have a map that sends identities $A = B$ to equivalences

$A \simeq B$, and the univalence axiom states that this map is an equivalence. Concretely, this means that properties of types are invariant under equivalence and that two types share the same properties whenever we have an equivalence between them.

Univalent foundations is especially interesting for the study of category theory. In category theory, objects are viewed up to isomorphism: whenever there is an isomorphism between two objects, they share the same categorical properties. This is known as the *principle of equivalence*, and this principle is made precise using *univalent categories*.

Univalent Categories. In univalent foundations, the "correct" notion of category is given by univalent categories. Given two objects x and y in a category, we have a map sending identities $x = y$ to isomorphisms $x \cong y$. In a univalent category, this mapping is required to be an equivalence: identities between objects are thus the same as isomorphisms. Hence, whenever two objects are isomorphic, they satisfy the same properties. Semantically, this is the correct notion of category, because in the simplicial set model, univalent categories correspond to set-theoretic categories [11].

There are several consequences of univalence for categories. For instance, we have a *structure identity principle* for univalent categories. This principle says that the identity type of two categories is equivalent to the type of adjoint equivalences between them [2]. As a consequence, one gets that whenever two categories are equivalent, then they have the same properties. Another consequence is that every essentially surjective fully faithful functor is an adjoint equivalence as well. Usually, one uses the axiom of choice to prove this principle, but if the domain is univalent, then one can constructively prove this fact. Finally, every category is weakly equivalent to a univalent one, which is called its *Rezk completion* [2].

While most categories that one encounters in practice are univalent (e.g., Eilenberg-Moore categories and functor categories), some are not. An example is given by the Kleisli category. Usually, the Kleisli category $\mathsf{K}(\mathsf{T})$ of a monad T on a category C is defined to be the category whose objects are objects of C and whose morphisms from x to y are morphisms $x \to \mathsf{T}\, y$ in C. However, this does not give rise to a univalent category in general. One can give an alternative presentation of the Kleisli category as a full subcategory $\mathsf{Kleisli}(\mathsf{T})$ of the Eilenberg-Moore category to obtain a univalent category [4]. To prove the desired theorems about $\mathsf{Kleisli}(\mathsf{T})$, one uses that it is the Rezk completion of $\mathsf{K}(\mathsf{T})$ [36].

Univalent Enriched Categories. In this paper, we develop enriched category theory in univalent foundations. More specifically, we define univalent enriched categories, and we prove analogous theorems for univalent enriched categories as for univalent categories. We show that univalent enriched categories satisfy a structure identity principle, that every essentially surjective fully faithful functor is an adjoint equivalence, and that every enriched category admits a Rezk completion. We also use these theorems to construct univalent enriched Kleisli categories.

Related work. While there are numerous libraries that contain a formalization of categories, enriched categories have gotten less attention. Several libraries, such as Agda categories [10] using Agda [22], mathlib [18] using Lean, and the category-theory library [38] in `Coq` [29], contain a couple of basic definitions. In `UniMath`, Satoshi Kura also formalized several basic concepts of enriched categories. However, none of the aforementioned formalizations consider much of the theory of enriched categories, and they do not consider univalent enriched categories. In addition, we use enrichments (Definition 2.2), while the other formalizations use the definition as given by Kelly [12]. Enrichments have been used in the setting of skew-enriched categories [5], and, with a slightly different definition, in the study of strong monads [20, Definition 5.1].

Formalization. The results in this paper are formalized in the `Coq` [29] proof assistant using the `UniMath` library [33]. We use the `UniMath` library in this work, because we frequently use notions from bicategory theory that have only been formalized in UniMath up to now. Definitions and theorems are accompanied with links to their corresponding identifier in the formalization, and these links are underlined. The tool `coqwc` reports the following number of lines of code in the formalization.

```
 spec    proof comments
17855     9242      520 total
```

Beside what is discussed in this paper, the formalization also contains (weighted) limits and colimits in enriched categories and models of the enriched effect calculus [7].

Contributions and Overview. The contributions of this paper are as follows.

- A construction of the bicategory of univalent enriched categories (Definition 2.6) and a proof that this bicategory is univalent (Theorem 2.7);
- a construction of the image factorization system of enriched categories (Construction 4.7);
- a proof that all fully faithful and essentially surjective enriched functors are adjoint equivalences (Theorem 4.8);
- a construction of the Rezk completion for enriched categories (Construction 5.4) and a proof of its universal property (Theorem 5.5);
- a construction of Kleisli objects (Construction 6.9) in the bicategory of univalent enriched categories.

In Section 3, we discuss numerous examples of enriched categories.

2 The Bicategory of Enriched Categories

In the remainder of this paper, we study univalent enriched categories, and in this section, we discuss a structure identity principle for univalent enriched categories, which says that identity of univalent enriched categories is the same as equivalence. Before we do so, we briefly recall *monoidal categories* to fix the notation for the remainder of the paper [3, 16].

▶ **Definition 2.1.** *A **monoidal category** consists of a category V together with*

- *an object $\mathbb{1}_{\mathsf{V}} : \mathsf{V}$;*
- *a bifunctor $- \otimes - : \mathsf{V} \times \mathsf{V} \to \mathsf{V}$;*
- *isomorphisms $\mathsf{l}_x : \mathbb{1}_{\mathsf{V}} \otimes x \to x$, $\mathsf{r}_x : x \otimes \mathbb{1}_V \to x$, and $\mathsf{a}_{x,y,z} : (x \otimes y) \otimes z \to x \otimes (y \otimes z)$;*

such that l_-, r_-, and $\mathsf{a}_{-,-,-}$ are natural and such that the triangle and pentagon laws hold.

*A **symmetric monoidal category** is a monoidal category V together with morphisms $\mathsf{s}_{x,y} : x \otimes y \to y \otimes x$, such that $\mathsf{s}_{x,y}$ and $\mathsf{s}_{y,x}$ are inverses, $\mathsf{s}_{-,-}$ is natural, and satisfies the hexagon law.*

*A **symmetric monoidal closed category** is a symmetric monoidal category V such that for every $x : \mathsf{V}$ the functor $x \otimes -$ has a right adjoint.*

If we have a symmetric monoidal closed category, then we have internal homs $x \multimap y$. We also have evaluation morphisms $\epsilon_{x,y} : (x \multimap y) \otimes x \to y$, and an internal lambda abstraction operation $\lambda(f) : x \to y \multimap z$ for every $f : x \otimes y \to z$.

Usually, a V-enriched category consists of a collection objects together with a hom-object $\mathcal{E}(x, y)$ in V for all x and y, such that we have appropriate identity morphisms and a composition operation [12]. Every enriched category $\mathcal{E}$ has an underlying category, which has the same collection of objects and whose morphisms from x to y are the same as morphisms $\mathbb{1} \to \mathcal{E}(x, y)$ in V.

However, we take a slightly different approach: we use *enrichments*. An enrichment for a category C consists of a hom-object $\mathcal{E}(x, y)$ in V for all x and y, such that we have the appropriate identity and composition morphisms and such that C is equivalent to the underlying category of the corresponding enriched category. As such, we view an enriched category as a category together with extra structure. This viewpoint also determines our notion of univalence for enriched categories: a univalent enriched category is an enriched category such that its underlying category is univalent. Note that our notion of univalence is similar to completeness for enriched ∞-categories [8]. Using enrichments, we can equivalently phrase univalent enriched categories as a univalent category together with an enrichment.

▶ **Definition 2.2.** *Let V be a monoidal category and let C be a category. A* **V-enrichment** *$\mathcal{E}$ for C consists of*

- *for all objects $x, y : \mathsf{C}$ an object $\mathcal{E}(x, y) : \mathsf{V}$;*
- *for every object $x : \mathsf{C}$ a morphism $\mathsf{id}^{\mathsf{e}}_{\mathcal{E}}(x) : \mathbb{1}_V \to \mathcal{E}(x, x)$;*
- *for all objects $x, y, z : \mathsf{C}$ a morphism $\mathsf{comp}_{\mathcal{E}}(x, y, z) : \mathcal{E}(y, z) \otimes \mathcal{E}(x, y) \to \mathcal{E}(x, z)$;*
- *for all morphisms $f : x \to y$ in C a morphism $\overrightarrow{f} : \mathbb{1}_V \to \mathcal{E}(x, y)$;*
- *for every morphism $f : \mathbb{1}_V \to \mathcal{E}(x, y)$ a morphism $\overleftarrow{f} : x \to y$ in C.*

In addition, we require that $\overleftarrow{\overrightarrow{f}} = f$ and $\overrightarrow{\overleftarrow{f}} = f$ and that the following diagrams commute.

$$\begin{array}{ccc} \mathbb{1} \otimes \mathcal{E}(x,y) & \xrightarrow{\mathsf{id}^{\mathsf{e}}(y)\otimes \mathrm{id}} & \mathcal{E}(y,y) \otimes \mathcal{E}(x,y) \\ & \searrow^{\mathsf{l}_{\mathcal{E}(x,y)}} & \downarrow{\scriptstyle \mathsf{comp}(x,y,y)} \\ & & \mathcal{E}(x,y) \end{array} \qquad \begin{array}{ccc} \mathcal{E}(x,y) \otimes \mathbb{1} & \xrightarrow{\mathrm{id}\otimes \mathsf{id}^{\mathsf{e}}(x)} & \mathcal{E}(x,y) \otimes \mathcal{E}(x,x) \\ & \searrow^{\mathsf{r}_{\mathcal{E}(x,y)}} & \downarrow{\scriptstyle \mathsf{comp}(x,x,y)} \\ & & \mathcal{E}(x,y) \end{array}$$

$$\begin{array}{ccc} (\mathcal{E}(y,z) \otimes \mathcal{E}(x,y)) \otimes \mathcal{E}(w,x) & \xrightarrow{\mathsf{a}} & \mathcal{E}(y,z) \otimes (\mathcal{E}(x,y) \otimes \mathcal{E}(w,x)) \\ \downarrow{\scriptstyle \mathsf{comp}(x,y,z)\otimes \mathrm{id}} & & \downarrow{\scriptstyle \mathrm{id}\otimes \mathsf{comp}(w,x,y)} \\ & & \mathcal{E}(y,z) \otimes \mathcal{E}(w,y) \\ & & \downarrow{\scriptstyle \mathsf{comp}(w,y,z)} \\ \mathcal{E}(x,z) \otimes \mathcal{E}(w,x) & \xrightarrow[\mathsf{comp}(w,x,z)]{} & \mathcal{E}(w,z) \end{array}$$

When it is clear from the context, we leave the arguments of comp and id^{e} implicit. In addition, note that the morphism id^{e} is redundant in Definition 2.2, since we have that $\mathsf{id}^{\mathsf{e}}_{\mathcal{E}}(x) = \overrightarrow{\mathrm{id}}$. However, we decided to keep id^{e} in the definition, because then it is slightly more convenient to prove Proposition 2.3.

In the remainder, we use the following operations for V-enrichments $\mathcal{E}$ for a category C. Given an object w and a morphism $f : x \to y$, we define f^{pre} as the following composition.

$$\mathcal{E}(w,x) \xrightarrow{\mathsf{l}^{-1}} \mathbb{1} \otimes \mathcal{E}(w,x) \xrightarrow{\overrightarrow{f}\otimes \mathrm{id}} \mathcal{E}(x,y) \otimes \mathcal{E}(w,x) \xrightarrow{\mathsf{comp}} \mathcal{E}(w,y)$$

For objects z and morphisms $f : x \to y$, we define f^{post} as the following composition.

$$\mathcal{E}(y,z) \xrightarrow{\mathsf{r}^{-1}} \mathcal{E}(y,z) \otimes \mathbb{1} \xrightarrow{\mathrm{id}\otimes \overrightarrow{f}} \mathcal{E}(y,z) \otimes \mathcal{E}(x,y) \xrightarrow{\mathsf{comp}} \mathcal{E}(x,z)$$

Note that a category together with a V-enrichment is the same as an enriched category as defined by Kelly [12].

▶ **Proposition 2.3.** *For a monoidal category V, the type of categories together with a V-enrichment is equivalent to the type of V-enriched categories.*

The reason why we use enrichments over the usual definition, is because it simplifies the proof of the structure identity principle for enriched categories. A structure identity principle is already present for univalent categories [2, Theorem 6.17], which can be reused directly if we define enriched categories as pairs of univalent categories together with an enrichment. However, if we would use the definition in [12] instead, then reusing this principle would be more cumbersome and thus the desired proof would be more involved.

To phrase the structure identity principle for enriched categories, we use *univalent bicategories*. More specifically, this principle for enriched categories is expressed by saying that the bicategory of enriched categories is univalent. We define the bicategory of enriched categories using *displayed bicategories* [1]. A displayed bicategory over a bicategory B represents structures and properties to be added to objects, 1-cells, and 2-cells in B. In our case, we define a displayed bicategory $\mathsf{dEnrichCat_V}$ over the bicategory $\mathsf{UnivCat}$ of univalent categories, and then $\mathsf{EnrichCat_V}$ is total bicategory of $\mathsf{dEnrichCat_V}$. The displayed objects over a univalent category C are V-enrichments for C, the displayed 1-cells over a functor are enrichments for functors, and the displayed 2-cells over a natural transformation are proofs that this transformation is enriched.

Note that from the machinery of displayed bicategories, we get a pseudofunctor $\mathsf{Und_V} : \mathsf{EnrichCat_V} \to \mathsf{UnivCat}$, which sends every enriched category to its underlying category. Using this pseudofunctor, we can understand an enrichment for C to be an object in the fiber of C along $\mathsf{Und_V}$.

▶ **Definition 2.4.** *Suppose that we have V-enrichments $\mathcal{E}_1$ and $\mathcal{E}_2$ for C_1 and C_2 respectively. A V-**enrichment** $\mathcal{F}$ for a functor $\mathsf{F} : \mathsf{C}_1 \to \mathsf{C}_2$ from $\mathcal{E}_1$ to $\mathcal{E}_2$ is a family of morphisms $\mathcal{F}(x, y) : \mathcal{E}_1(x, y) \to \mathcal{E}_2(\mathsf{F}\, x, \mathsf{F}\, y)$ such that the following diagrams commute*

$$\begin{array}{ccc} \mathbb{1} & \xrightarrow{\mathsf{id}^{\mathsf{e}}} & \mathcal{E}_1(x,x) \\ & \searrow^{\mathsf{id}^{\mathsf{e}}} & \downarrow{\scriptstyle \mathcal{F}(x,x)} \\ & & \mathcal{E}_2(\mathsf{F}\, x, \mathsf{F}\, x) \end{array} \qquad \begin{array}{ccc} \mathcal{E}_1(y,z) \otimes \mathcal{E}_1(x,y) & \xrightarrow{\mathsf{comp}} & \mathcal{E}_1(x,z) \\ {\scriptstyle \mathcal{F}(y,z)\otimes\mathcal{F}(x,y)}\downarrow & & \downarrow{\scriptstyle \mathcal{F}(x,z)} \\ \mathcal{E}_2(\mathsf{F}\, y, \mathsf{F}\, z) \otimes \mathcal{E}_2(\mathsf{F}\, x, \mathsf{F}\, y) & \xrightarrow[\mathsf{comp}]{} & \mathcal{E}_2(\mathsf{F}\, x, \mathsf{F}\, z) \end{array}$$

In addition, we require that $\overrightarrow{\mathsf{F}\, f} = \overrightarrow{f} \cdot \mathcal{F}(x, y)$.

▶ **Definition 2.5.** *Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be V-enrichments for functors $\mathsf{F}_1, \mathsf{F}_2 : \mathsf{C}_1 \to \mathsf{C}_2$ from $\mathcal{E}_1$ to $\mathcal{E}_2$. A natural transformation $\tau : F_1 \Rightarrow F_2$ is called V-**enriched** whenever the following diagram commutes*

$$\begin{array}{ccccc} & & \mathcal{E}_1(x,y) \otimes \mathbb{1} & \xrightarrow{\mathcal{F}_2 \otimes \overrightarrow{\tau\, x}} & \mathcal{E}_2(\mathsf{F}_2\, x, \mathsf{F}_2\, y) \otimes \mathcal{E}_2(\mathsf{F}_1\, x, \mathsf{F}_2\, x) \\ & \nearrow^{\mathsf{r}^{-1}} & & & \searrow^{\mathsf{comp}} \\ \mathcal{E}_1(x,y) & & & & \mathcal{E}_2(\mathsf{F}_1\, x, \mathsf{F}_2\, y) \\ & \searrow_{\mathsf{l}^{-1}} & & & \nearrow_{\mathsf{comp}} \\ & & \mathbb{1} \otimes \mathcal{E}_1(x,y) & \xrightarrow[\overrightarrow{\tau\, y} \otimes \mathcal{F}_1]{} & \mathcal{E}_2(\mathsf{F}_1\, y, \mathsf{F}_2\, y) \otimes \mathcal{E}_2(\mathsf{F}_1\, x, \mathsf{F}_1\, y) \end{array}$$

Note that the condition for V-enriched natural transformations can equivalently formulated by saying that the following diagram commutes.

$$\begin{array}{ccc} \mathcal{E}_1(x,y) & \xrightarrow{\mathcal{F}_1} & \mathcal{E}_2(\mathsf{F}_1\, x, \mathsf{F}_1\, y) \\ {\scriptstyle \mathcal{F}_2}\downarrow & & \downarrow{\scriptstyle (\tau\, y)^{\mathsf{post}}} \\ \mathcal{E}_2(\mathsf{F}_2\, x, \mathsf{F}_2\, y) & \xrightarrow[(\tau\, x)^{\mathsf{pre}}]{} & \mathcal{E}_2(\mathsf{F}_1\, x, \mathsf{F}_2\, y) \end{array} \tag{1}$$

Now we have everything in place to define the bicategory of enriched categories.

▶ **Definition 2.6.** *Let* V *be a monoidal category. We define the* ***displayed bicategory*** $\mathsf{dEnrichCat_V}$ ***of enrichments*** *over* $\mathsf{UnivCat}$ *as follows.*

- *The displayed objects over a category* C *are* V*-enrichments for* C*;*
- *the displayed 1-cells over a functor* $\mathsf{F} : \mathsf{C}_1 \to \mathsf{C}_2$ *from* $\mathcal{E}_1$ *to* $\mathcal{E}_2$ *are* V*-enrichments for* F *from* C_1 *to* C_2*;*
- *the displayed 2-cells over a natural transformation* $\tau : \mathsf{F}_1 \Rightarrow \mathsf{F}_2$ *from* $\mathcal{F}_1$ *to* $\mathcal{F}_2$ *are proofs that* τ *is* V*-enriched.*

The ***bicategory of enriched categories*** *is defined to be the total bicategory of* $\mathsf{dEnrichCat_V}$*, and we denote it by* $\mathsf{EnrichCat_V}$*. Its objects are* ***univalent*** V***-enriched categories****, and we call the 1-cells and 2-cells of* $\mathsf{EnrichCat_V}$ ***enriched functors*** *and* ***enriched transformations*** *respectively.*

Note that by construction a univalent V-enriched category is the same as a univalent category together with a V-enrichment. In addition, our univalence condition for V-enriched categories has no local variant in contrast to univalence for bicategories [1], since we only look at enrichments over monoidal 1-categories.

To show that Definition 2.6 actually gives rise to a displayed bicategory, one also needs to construct enrichments for the identity and composition, and one needs to prove that the identity transformation is enriched and that enriched transformations are preserved under composition and whiskering. Details on this construction are left to the formalization.

▶ **Theorem 2.7.** *If* V *is a univalent monoidal category, then the bicategory* $\mathsf{EnrichCat_V}$ *is univalent.*

▶ **Proposition 2.8.** *Let* $\tau : \mathcal{F}_1 \Rightarrow \mathcal{F}_2$ *be a 2-cell in* $\mathsf{EnrichCat_V}$*. Then* τ *is invertible if the underlying natural transformation of* τ *is a natural isomorphism.*

3 Examples of Enriched Categories

Before we continue our study of univalent enriched categories, we first look at numerous examples of enrichments that we use in the remainder of this paper. In Section 3.2, we characterize enrichments over a large class of structures.

3.1 General Examples

▶ **Example 3.1.** *Let* V *be a symmetric monoidal closed category. We define a* V*-enrichment for* V*, which we call the* ***self-enrichment*** *and denote by* $\mathsf{self}(\mathsf{V})$*, as follows.*

- *We define* $\mathsf{self}(\mathsf{V})(x, y)$ *to be* $x \multimap y$*.*
- *The enriched identity* $\mathsf{id}^{\mathsf{e}}(x) : \mathbb{1} \to x \multimap x$ *is defined to be* $\lambda(\mathsf{l}_x)$*.*
- *The composition* $\mathsf{comp}(x, y, z) : y \multimap z \otimes x \multimap y \to x \multimap z$ *is the exponential transpose of the following composition of morphisms.*

$$((y \multimap z) \otimes (x \multimap y)) \otimes x \xrightarrow{\mathsf{a}} (y \multimap z) \otimes ((x \multimap y) \otimes x) \xrightarrow{\mathrm{id} \otimes \epsilon} (y \multimap z) \otimes y \xrightarrow{\epsilon} z$$

- *Given* $f : x \to y$*, we define* $\overrightarrow{f} : \mathbb{1} \to x \multimap y$ *to be* $\lambda(\mathsf{l}_x \cdot f)$*.*
- *For* $f : \mathbb{1} \to x \multimap y$*, we define* $\overleftarrow{f}$ *to be the following composition of morphisms.*

$$x \xrightarrow{\mathsf{l}_x^{-1}} \mathbb{1} \otimes x \xrightarrow{f \otimes \mathrm{id}_x} (x \multimap y) \otimes x \xrightarrow{\epsilon} y$$

If we assume that V *is univalent, then* $\mathsf{self}(\mathsf{V})$ *is a univalent enriched category.*

▶ **Example 3.2.** *Let* C *be a category together with a* V*-enrichment* $\mathcal{E}$*, and let* P *be a predicate on the objects of* C*. From all of this, we obtain a* V***-enrichment*** $\mathsf{FSub}_{\mathsf{e}}(\mathsf{P})$ ***for the full subcategory*** $\mathsf{FSub}(\mathsf{P})$*, such that* $\mathsf{FSub}_{\mathsf{e}}(\mathsf{P})(x, y) \coloneqq \mathcal{E}(x, y)$*. If* C *is univalent, then the full subcategory of* C *is also univalent, and in that case, this construction gives rise to a univalent enriched category.*

▶ **Example 3.3.** *Suppose that* V *is a symmetric monoidal category, and let* C *be a category together with a* V*-enrichment* $\mathcal{E}$*. We define the* V*-enrichment* $\mathcal{E}^{\mathsf{op}}$*, called the* ***opposite enrichment****, for* C^{op} *as follows.*

- $\mathcal{E}^{\mathsf{op}}(x, y) \coloneqq \mathcal{E}(y, x)$*;*
- $\mathsf{id}^{\mathsf{e}}_{\mathcal{E}^{\mathsf{op}}}(x) \coloneqq \mathsf{id}^{\mathsf{e}}_{\mathcal{E}}(x)$*;*
- $\mathsf{comp}_{\mathcal{E}^{\mathsf{op}}}(x, y, z) \coloneqq \mathcal{E}(z, y) \otimes \mathcal{E}(y, x) \xrightarrow{\mathsf{s}} \mathcal{E}(y, x) \otimes \mathcal{E}(z, y) \xrightarrow{\mathsf{comp}} \mathcal{E}(z, x)$

The operations $\overrightarrow{f}$ *and* $\overleftarrow{f}$ *in* $\mathcal{E}^{\mathsf{op}}$ *are inherited from* $\mathcal{E}$*. In addition,* $\mathcal{E}^{\mathsf{op}}$ *gives rise to a univalent enriched category if* C *is univalent.*

In fact, using Example 3.3 one can construct a duality involution on $\mathsf{EnrichCat}_{\mathsf{V}}$.

▶ **Example 3.4.** *Suppose that* V *is a symmetric monoidal category that has equalizers, and suppose that we have two enriched functors* $\mathcal{F}_1, \mathcal{F}_2 : \mathcal{E}_1 \to \mathcal{E}_2$*. We have the category* $\mathsf{Dialg}(\mathcal{F}_1, \mathcal{F}_2)$ *of dialgebras whose objects are pairs* (x, f) *consisting of an object* $x : \mathcal{E}_1$ *together with a morphism* $f : \mathcal{F}_1\, x \to \mathcal{F}_2\, x$*. Morphisms from* (x, f) *to* (y, g) *are morphisms* $h : x \to y$ *such that the following diagram commutes.*

$$\begin{array}{ccc} \mathcal{F}_1\, x & \xrightarrow{\mathcal{F}_1\, h} & \mathcal{F}_1\, y \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle g} \\ \mathcal{F}_2\, x & \xrightarrow[\mathcal{F}_2\, h]{} & \mathcal{F}_2\, y \end{array}$$

We define a V***-enrichment*** $\mathsf{Dialg}_{\mathsf{e}}(\mathcal{F}_1, \mathcal{F}_2)$ ***for*** $\mathsf{Dialg}(\mathcal{F}_1, \mathcal{F}_2)$*. Suppose that we have objects* (x, f) *and* (y, g) *in* $\mathsf{Dialg}(\mathcal{F}_1, \mathcal{F}_2)$*. We define the object* $\mathsf{Dialg}_{\mathsf{e}}(\mathcal{F}_1, \mathcal{F}_2)((x, f), (y, g))$ *as the equalizer of the following diagram.*

$$\mathsf{Dialg}_{\mathsf{e}}(\mathcal{F}_1, \mathcal{F}_2)((x, f), (y, g)) \hookrightarrow \mathcal{E}_1(x, y) \rightrightarrows \mathcal{E}_2(\mathcal{F}_1\, x, \mathcal{F}_1\, y) \rightrightarrows \mathcal{E}_2(\mathcal{F}_1\, x, \mathcal{F}_2\, y)$$

To define the enriched identity and composition morphisms, one uses the universal property of equalizers. If C *is univalent, then so is the category of dialgebras, and in that case,* $\mathsf{Dialg}_{\mathsf{e}}(\mathcal{F}_1, \mathcal{F}_2)$ *is a univalent enriched category.*

Using Example 3.4, one can show that the bicategory $\mathsf{EnrichCat}_{\mathsf{V}}$ has inserters.

▶ **Example 3.5.** *Let* V *be a complete symmetric monoidal category, and suppose that we have enriched categories* $\mathcal{E}_1$ *and* $\mathcal{E}_2$*. Note that we have a category* $[\mathcal{E}_1, \mathcal{E}_2]$ *whose objects are given by enriched functors from* $\mathcal{E}_1$ *to* $\mathcal{E}_2$*, and whose morphisms are given by enriched natural transformations. We define a* V***-enrichment*** $\mathsf{EFunctor}(\mathcal{E}_1, \mathcal{E}_2)$ ***for*** $[\mathcal{E}_1, \mathcal{E}_2]$ *as the equalizer of the morphisms displayed below.*

$$\textstyle\prod_{x:\mathcal{E}_1} \mathcal{E}_2(\mathcal{F}_1\, x, \mathcal{F}_2\, x) \underset{g}{\overset{f}{\rightrightarrows}} \prod_{x,y:\mathcal{E}_1} \mathcal{E}_1(x, y) \multimap \mathcal{E}_2(\mathcal{F}_1\, x, \mathcal{F}_2\, y)$$

Here f is defined to be the following composition of morphisms

$$\prod_{x:\mathcal{E}_1} \mathcal{E}_2(\mathcal{F}_1\, x, \mathcal{F}_2\, x) \xrightarrow{\pi_y} \mathcal{E}_2(\mathcal{F}_1\, y, \mathcal{F}_2\, y) \xrightarrow{\varphi} \mathcal{E}_1(x,y) \multimap \mathcal{E}_2(\mathcal{F}_1\, x, \mathcal{F}_2\, y)$$

where φ is the exponential transpose of

$$\mathcal{E}_2(\mathcal{F}_1\, y, \mathcal{F}_2\, y) \otimes \mathcal{E}_1(x,y) \xrightarrow{\mathrm{id} \otimes \mathcal{F}_1} \mathcal{E}_2(\mathcal{F}_1\, y, \mathcal{F}_2\, y) \otimes \mathcal{E}_2(\mathcal{F}_1\, x, \mathcal{F}_1\, y) \xrightarrow{\mathsf{comp}} \mathcal{E}_2(\mathcal{F}_1\, x, \mathcal{F}_2\, y).$$

We define g analogously. The fact that $[\mathcal{E}_1, \mathcal{E}_2]$ is univalent, follows from the fact that $\mathsf{EnrichCat}_{\mathsf{V}}$ *is univalent (Theorem 2.7).*

Inspired by Example 3.5, we can refine Example 3.1. More specifically, given a small category C, we define a V-enrichment for the functor category from C to V. The construction is analogous to Example 3.5, and details can be found in the formalization.

Finally, we look at the change of base operation for enriched categories, and for this operation, a subtlety arises. Given a lax monoidal functor $\mathsf{F} : \mathsf{V}_1 \to \mathsf{V}_2$, our goal is to define a pseudofunctor $\mathsf{EnrichCat}_{\mathsf{V}_1} \to \mathsf{EnrichCat}_{\mathsf{V}_2}$. On objects, this operation acts as follows: given a univalent category C together with an enrichment $\mathcal{E}$, then we get an enriched category $\mathsf{F}^*(\mathcal{E})$ whose objects are objects in C and such that $\mathsf{F}^*(\mathcal{E})(x,y) = \mathsf{F}\,(\mathcal{E}(x,y))$. However, the underlying category of this enriched category is **not** necessarily univalent. For instance, if we take F to be the unique monoidal functor from Set to the terminal category, then the underlying category of $\mathsf{F}^*(\mathcal{E})$ would have sets as objects, and inhabitants of the unit type as the morphisms. For this reason, we add a restriction to F in order to define the change of base of enriched categories.

▶ **Definition 3.6.** *Let $\mathsf{F} : \mathsf{V}_1 \to \mathsf{V}_2$ by a lax monoidal functor. We say that F* ***preserves underlying categories*** *if for all $x : \mathsf{V}_1$ the function that sends morphisms $f : \mathbb{1}_{\mathsf{V}_1} \to x$ to $\mathbb{1}_{\mathsf{V}_2} \xrightarrow{\epsilon_{\mathsf{F}}} \mathsf{F}\,\mathbb{1}_{\mathsf{V}_1} \xrightarrow{\mathsf{F}\,f} \mathsf{F}\,x$ is an equivalence of types. If we have $f : \mathbb{1}_{\mathsf{V}_2} \to \mathsf{F}\,x$, then we denote the action of the inverse by $\zeta_{\mathsf{F}}(f)$.*

The requirement in Definition 3.6 says that the underlying category is preseerved by change of base along F. With this additional assumption, we define the change of base of enriched categories.

▶ **Example 3.7.** *Let $\mathsf{F} : \mathsf{V}_1 \to \mathsf{V}_2$ by a lax monoidal functor that preserves underlying categories, and let C be a category together with a V-enrichment $\mathcal{E}$. We define the* ***change-of-base enrichment*** *$\mathsf{F}^*(\mathcal{E})$ for C as follows. The hom-object $\mathsf{F}^*(\mathcal{E})(x,y)$ is defined to be $\mathsf{F}\,(\mathcal{E}(x,y))$, and the enriched identity $\mathsf{id}^{\mathsf{e}}(x)$ is defined as the composition*

$$\mathbb{1}_{\mathsf{V}_2} \xrightarrow{\epsilon_{\mathsf{F}}} \mathsf{F}\,\mathbb{1}_{\mathsf{V}_1} \xrightarrow{\mathsf{F}\,(\mathsf{id}^{\mathsf{e}}(x))} \mathsf{F}\,(\mathcal{E}(x,x))$$

Composition is defined similarly.

$$\mathsf{F}\,(\mathcal{E}(y,z)) \otimes \mathsf{F}\,(\mathcal{E}(x,y)) \xrightarrow{\mu_{\mathsf{F}}} \mathsf{F}\,(\mathcal{E}(y,z) \otimes \mathcal{E}(x,y)) \xrightarrow{\mathsf{F}\,\mathsf{comp}} \mathsf{F}\,(\mathcal{E}(x,y))$$

If we have a morphism $f : x \to y$, then we define $\overrightarrow{f}$ to be

$$\mathbb{1}_{\mathsf{V}_2} \xrightarrow{\epsilon_{\mathsf{F}}} \mathsf{F}\,\mathbb{1}_{\mathsf{V}_1} \xrightarrow{\mathsf{F}\,(\overrightarrow{f})} \mathsf{F}\,(\mathcal{E}(x,x)).$$

Finally, for a morphism $f : \mathbb{1}_{\mathsf{V}_2} \to \mathsf{F}\,(\mathcal{E}(x,x))$, we define $\overleftarrow{f}$ to be $\overleftarrow{\zeta_{\mathsf{F}}(f)}$. Note that here we use that F preserves underlying categories. In addition, if we assume that C is univalent, then we get a univalent enriched category $\mathsf{F}^*(\mathcal{E})$.

3.2 Enrichments over Structures

Next we characterize two classes of enrichments. First, we characterize enrichments for the category Set of sets equipped with its cartesian monoidal structure.

▶ **Proposition 3.8.** *Let* C *be a category. The type of* Set*-enrichments for* C *is contractible.*

From Proposition 3.8, we can conclude that the type of categories is equivalent to the type of Set-enriched categories. Second, we characterize enrichments for structured sets with a cartesian monoidal structure. To do so, we first define a general notion of structured sets.

▶ **Definition 3.9.** *A* ***cartesian notion of structure*** $\mathcal{S}$ *consists of*

- *a set* $\mathsf{P}_{\mathcal{S}}\,\mathsf{X}$ *of structures on* X *for every set* X*;*
- *a proposition* $\mathsf{H}_{(\mathsf{p}_\mathsf{X},\mathsf{p}_\mathsf{Y})}(f)$ *which represents that* f *is a structure preserving map from* p_X *to* p_Y*, for all functions* $f : \mathsf{X} \to \mathsf{X}$ *and structures* $\mathsf{p}_\mathsf{X} : \mathsf{P}_{\mathcal{S}}\,\mathsf{X}$ *and* $\mathsf{p}_\mathsf{Y} : \mathsf{P}_{\mathcal{S}}\,\mathsf{Y}$*;*
- *an inhabitant* $\mathsf{p}_{\mathsf{unit}} : \mathsf{P}_{\mathcal{S}}\,\mathsf{unit}$*;*
- *a structure* $\mathsf{p}_\mathsf{X} \times \mathsf{p}_\mathsf{Y} : \mathsf{P}_{\mathcal{S}}\,(\mathsf{X} \times \mathsf{Y})$ *for all* $\mathsf{p}_\mathsf{X} : \mathsf{P}_{\mathcal{S}}\,\mathsf{X}$ *and* $\mathsf{p}_\mathsf{Y} : \mathsf{P}_{\mathcal{S}}\,\mathsf{Y}$*,.*

This data is required to satisfy the following axioms.

- *For every set* X *and structure* $\mathsf{p}_\mathsf{X} : \mathsf{P}_{\mathcal{S}}\,\mathsf{X}$*, we have* $\mathsf{H}_{(\mathsf{p}_\mathsf{X},\mathsf{p}_\mathsf{X})}(\mathrm{id}_\mathsf{X})$*;*
- *for all functions* $f : \mathsf{X} \to \mathsf{Y}$ *and* $g : \mathsf{Y} \to \mathsf{Z}$ *such that* $\mathsf{H}_{(\mathsf{P}_{\mathcal{S}}\,\mathsf{X},\mathsf{P}_{\mathcal{S}}\,\mathsf{Y})}(f)$ *and* $\mathsf{H}_{(\mathsf{P}_{\mathcal{S}}\,\mathsf{Y},\mathsf{P}_{\mathcal{S}}\,\mathsf{Z})}(g)$*, we have* $\mathsf{H}_{(\mathsf{P}_{\mathcal{S}}\,\mathsf{X},\mathsf{P}_{\mathcal{S}}\,\mathsf{Z})}(g \circ f)$*;*
- *given structures* $\mathsf{p}_\mathsf{X}, \mathsf{p}'_\mathsf{X} : \mathsf{P}_{\mathcal{S}}\,\mathsf{X}$ *such that* $\mathsf{H}_{(\mathsf{p}_\mathsf{X},\mathsf{p}_{\mathsf{X}'})}(\mathrm{id}_\mathsf{X})$ *and* $\mathsf{H}_{(\mathsf{p}_{\mathsf{X}'},\mathsf{p}_\mathsf{X})}(\mathrm{id}_\mathsf{X})$*, we have* $\mathsf{p}_\mathsf{X} = \mathsf{p}'_\mathsf{X}$*;*
- *given a structure* $\mathsf{p}_\mathsf{X} : \mathsf{P}_{\mathcal{S}}\,\mathsf{X}$ *on a set* X*, we have* $\mathsf{H}_{(\mathsf{p}_\mathsf{X},\mathsf{p}_{\mathsf{unit}})}(\lambda(x : \mathsf{X}).\mathsf{tt})$ *where* tt *is the unique element of* unit*;*
- *given structures* $\mathsf{p}_\mathsf{X} : \mathsf{P}_{\mathcal{S}}\,\mathsf{X}$ *and* $\mathsf{p}_\mathsf{Y} : \mathsf{P}_{\mathcal{S}}\,\mathsf{Y}$ *on sets* X *and* Y *respectively, we have* $\mathsf{H}_{(\mathsf{p}_\mathsf{X}\times\mathsf{p}_\mathsf{Y},\mathsf{p}_\mathsf{X})}(\pi_1)$ *and* $\mathsf{H}_{(\mathsf{p}_\mathsf{X}\times\mathsf{p}_\mathsf{Y},\mathsf{p}_\mathsf{Y})}(\pi_2)$*;*
- *for all functions* $f : \mathsf{X} \to \mathsf{Y}$ *and* $g : \mathsf{X} \to \mathsf{Z}$ *such that* $\mathsf{H}_{(\mathsf{P}_{\mathcal{S}}\,\mathsf{X},\mathsf{P}_{\mathcal{S}}\,\mathsf{Y})}(f)$ *and* $\mathsf{H}_{(\mathsf{P}_{\mathcal{S}}\,\mathsf{X},\mathsf{P}_{\mathcal{S}}\,\mathsf{Z})}(g)$*, we have* $\mathsf{H}_{(\mathsf{P}_{\mathcal{S}}\,\mathsf{X},\mathsf{p}_\mathsf{Y}\times\mathsf{p}_\mathsf{Z})}(\lambda(x : \mathsf{X}).(f\ x, g\ x))$*.*

Note that Definition 3.9 is extension of standard notions of structures defined in [31, Definition 9.8.1]: the added data and axioms guarantee that the resulting category has binary products and a terminal object.

▶ **Problem 3.10.** *Given a cartesian notion of structure* $\mathcal{S}$*, to construct a univalent cartesian category* $\mathsf{Str}(\mathcal{S})$*.*

▶ **Construction 3.11** (for Problem 3.10)**.** In [31, Section 9.8], it is shown how every standard notion of structure gives rise to a univalent category. The terminal object is given by $(\mathsf{unit}, \mathsf{p}_{\mathsf{unit}})$, and the product of $(\mathsf{X}, \mathsf{p}_\mathsf{X})$ and $(\mathsf{Y}, \mathsf{p}_\mathsf{Y})$ is given by $(\mathsf{X} \times \mathsf{Y}, \mathsf{p}_\mathsf{X} \times \mathsf{p}_\mathsf{Y})$. ⌟

▶ **Proposition 3.12.** *Let* C *be a category and let* $\mathcal{S}$ *be a cartesian notion of structure. Then the type of* $\mathsf{Str}(\mathcal{S})$*-enrichments for* C *is equivalent to a structure* $\mathsf{hom}_{(x,y)} : \mathsf{P}_{\mathcal{S}}\,(x \to y)$ *for all objects* $x, y : \mathsf{C}$ *such that for all* $x, y, z : \mathsf{C}$ *we have* $\mathsf{H}_{(\mathsf{hom}_{(y,z)}\times\mathsf{hom}_{(x,y)},\mathsf{hom}_{(x,z)})}(\lambda f.\pi_2\ f \cdot \pi_1\ f)$*.*

As such, to give a $\mathsf{Str}(\mathcal{S})$-enrichment for C one needs to endow every hom-set of C with an $\mathcal{S}$-structure such that the composition operation is a structure preserving map.

▶ **Example 3.13.** *We have a cartesian notion of structure* DCPO *of* ***directed complete partial orders structures*** *(DCPOs) such that* $\mathsf{P}_{\mathsf{DCPO}}\,\mathsf{X}$ *is the set of DCPOs on* X *and such that* $\mathsf{H}_{(\mathsf{p}_\mathsf{X},\mathsf{p}_\mathsf{Y})}(f)$ *expresses that* f *is a Scott continuous map. As such, a* DCPO*-enriched category is given by a category whose hom-sets are directed complete partial orders, and whose composition operation is a Scott-continuous map.*

We also have a cartesian notion of structure DCPPO *of* ***pointed directed complete partial orders structures*** *(DCPPOs) such that* $\mathsf{P}_{\mathsf{DCPPO}}\,\mathsf{X}$ *is the set of DCPPOs on* X *and such that* $\mathsf{H}_{(\mathsf{px},\mathsf{py})}(f)$ *expresses that* f *is a Scott continuous map. Hence,* DCPPO*-enriched categories are categories whose hom-sets are pointed directed complete partial orders, and whose composition operation is a Scott-continuous map.*

▶ **Remark 3.14.** *In Example 3.13, we defined a cartesian notion of structure by* pointed *DCPOs and Scott continuous maps without requiring these maps to be strict. For pointed DCPOs and strict Scott continuous maps, one can also define such a structure. However, in applications, one is often interested in a different monoidal structure for pointed DCPOs with strict maps, namely the one given by the smash product. Note that one can construct this symmetric monoidal category constructively [30, Theorem 2.9.1].*

The formalization contains a further extension of Definition 3.9, called a ***structure supporting smash products****, and a proof that every such structure gives rise to a symmetric monoidal closed category. For such structures, the smash product is constructed a quotient of types, and one can instantiate this notion using pointed sets and pointed partial orders.*

4 Image Factorization

We continue our study of univalent enriched categories by proving that every essentially surjective and fully faithful (enriched) functor is an adjoint equivalence. Classically, one would use the axiom of choice to prove this fact: to define the inverse, one needs to pick preimages and those are only guaranteed to be unique up to isomorphism. One can give a constructive proof of this fact if one assumes that the domain of the functor in question is univalent.

The way we approach this result, is via *orthogonal factorization systems* in bicategories. More specifically, we show that the essentially surjective and the fully faithful enriched functors form an orthogonal factorization system [15, Lemma 4.3.5]. From this fact, one directly obtains that every essentially surjective and fully faithful functor is an adjoint equivalence. The proof is similar to how in orthogonal factorization systems in categories the intersection of the left and right class of maps are precisely the isomorphisms.

We start by defining *orthogonal* maps in bicategories.

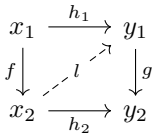
▶ **Definition 4.1.** *Let* B *be a bicategory and let* $f : x_1 \to x_2$ *and* $g : y_1 \to y_2$ *be 1-cells. Then we say that* f *is* ***orthogonal*** *to* g*, written* $f \perp g$*, if the following diagram of categories is a weak pullback in the bicategory of categories.*

$$\begin{array}{ccc} \underline{\mathsf{B}}(x_2, y_1) & \xrightarrow{f^{\mathsf{pre}}} & \underline{\mathsf{B}}(x_1, y_1) \\ {\scriptstyle g^{\mathsf{post}}}\downarrow & & \downarrow{\scriptstyle g^{\mathsf{post}}} \\ \underline{\mathsf{B}}(x_2, y_2) & \xrightarrow[f^{\mathsf{pre}}]{} & \underline{\mathsf{B}}(x_1, y_2) \end{array}$$

where the functors f^{pre} *and* g^{post} *are given by precomposition with* f *and postcomposition with* g *respectively.*

Let us reflect on Definition 4.1. Weak pullbacks of categories are given by iso-comma categories. The objects in the iso-comma category $\mathsf{F}/_{\cong}\mathsf{G}$ of functors $\mathsf{F} : \mathsf{C}_1 \to \mathsf{C}_3$ and $\mathsf{G} : \mathsf{C}_2 \to \mathsf{C}_3$ are given by triples (x, y, f) of objects $x : \mathsf{C}_1$ and $y : \mathsf{C}_2$ together with an isomorphism $f : \mathsf{F}\,x \cong \mathsf{G}\,y$. Note that we have a functor $\mathsf{O}_{(f,g)} : \underline{\mathsf{B}}(x_2, y_1) \to f^{\mathsf{pre}}/_{\cong}g^{\mathsf{post}}$. The functor $\mathsf{O}_{(f,g)}$ maps 1-cells $h : x_2 \to y_1$ to the triple $(h \cdot g, f \cdot h, \alpha_{f,h,g})$ where α is the

associator of B. Orthogonality can equivalently be phrased by saying that the functor $\mathsf{O}_{(f,g)}$ is an adjoint equivalence. Essential surjectivity of $\mathsf{O}_{(f,g)}$ says that every square has a diagonal filler as follows.



More concretely, given the diagram above, there is a lift $l : x_2 \to y_1$ making the two triangles commute up to invertible 2-cell. Fully faithfulness of $\mathsf{O}_{(f,g)}$ says that whenever we have two lifts $l_1, l_2 : x_2 \to y_1$ together with 2-cells $\tau_1 : l_1 \cdot g \Rightarrow l_2 \cdot g$ and $\tau_2 : f \cdot l_1 \Rightarrow f \cdot l_2$, we have a unique 2-cell $\zeta : l_1 \Rightarrow l_2$ such that $\zeta \rhd g = \tau_1$ and $f \lhd \zeta = \tau_2$.

▶ **Definition 4.2.** *Let B be a bicategory. An **orthogonal factorization system** on B consists of two classes of maps, which we denote by $\mathcal{L}$ and $\mathcal{R}$, such that*

- *$\mathcal{L}$ and $\mathcal{R}$ are closed under invertible 2-cells;*
- *for all 1-cells f and g such that $\mathcal{L}\, f$ and $\mathcal{R}\, g$, we have $f \perp g$;*
- *for every 1-cell f, we have a factorization $f \cong l \cdot r$ such that $\mathcal{L}\, l$ and $\mathcal{R}\, r$.*

In this section, we are interested in a particular factorization system on $\mathsf{EnrichedCat}_{\mathsf{V}}$, which is given by the fully faithful and the essentially surjective enriched functors.

▶ **Definition 4.3.** *Let $\mathcal{F} : \mathcal{E}_1 \to \mathcal{E}_2$ be an enriched functor.*

- *We say that $\mathcal{F}$ is **fully faithful** if for all objects $x, y : \mathcal{E}_1$ the morphism $\mathcal{F}(x, y)$ is an isomorphism.*
- *We say that $\mathcal{F}$ is **essentially surjective** if its underlying functor is essentially surjective. That is to say, for all $y : \mathcal{E}_2$ we have an inhabitant of $\|\sum(x : \mathcal{E}_1), \mathcal{F}\, x \cong y\|$.*
- *We say that $\mathcal{F}$ is a **weak equivalence** if $\mathcal{F}$ is both fully faithful and essentially surjective.*

Every enriched functor can be factorized as an essentially surjective functor followed by a fully faithful functor by taking the *full image*.

▶ **Example 4.4.** *Let $\mathcal{F} : \mathcal{E}_1 \to \mathcal{E}_2$ be an enriched functor. We define a predicate P on the objects of $\mathcal{E}_2$ such that $\mathsf{P}\, y :\equiv \|\sum(x : \mathcal{E}_1), \mathcal{F}\, x \cong y\|$. The **full image** $\mathsf{Im}(\mathcal{F})$ of $\mathcal{F}$ is defined to be the full subcategory of $\mathcal{E}_2$ with respect to P.*

▶ **Proposition 4.5.** *Suppose that we have univalent enriched categories $\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3$, and $\mathcal{E}_4$. If we have enriched functors $\mathcal{F} : \mathcal{E}_1 \to \mathcal{E}_2$ and $\mathcal{G} : \mathcal{E}_3 \to \mathcal{E}_4$ such that $\mathcal{F}$ is essentially surjective and $\mathcal{G}$ is fully faithful, then $\mathcal{F} \perp \mathcal{G}$.*

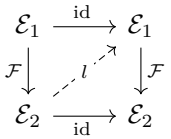
▶ **Problem 4.6.** *To construct an orthogonal factorization system on $\mathsf{EnrichedCat}_{\mathsf{V}}$.*

▶ **Construction 4.7** (for Problem 4.6)**.** The classes $\mathcal{L}$ and $\mathcal{R}$ are given by the essentially surjective and the fully faithful enriched functors respectively. The desired factorization is given by Example 4.4, and the proof of orthogonality is given in Proposition 4.5. It remains to show that essentially surjective and fully faithful enriched functors are closed under enriched natural isomorphisms, and the details for that is given in the formalization. ⌟

From this factorization system, we directly obtain that weak equivalence are actually adjoint equivalences.

▶ **Theorem 4.8.** *Every fully faithful and essentially surjective enriched functor $\mathcal{F} : \mathcal{E}_1 \to \mathcal{E}_2$ is an adjoint equivalence.*

Proof. Suppose that $\mathcal{F} : \mathcal{E}_1 \to \mathcal{E}_2$ is fully faithful and essentially surjective. Consider the following diagram



Due to the orthogonality of fully faithful and essentially surjective morphisms, this diagram has a lift l such that both triangles commute up to invertible 2-cell. From this, we get that f is an equivalence, and since equivalences can be refined to adjoint equivalences, f is an adjoint equivalence. ◀

▶ **Remark 4.9.** *For the proof of Theorem 4.8, we require both the domain and codomain of $\mathcal{F}$ to be univalent. This restriction is a consequence of the bicategorical machinery, because we phrase everything in the bicategory* $\mathsf{EnrichCat}_{\mathsf{V}}$ *whose objects are univalent enriched categories. In the case that only the domain of $\mathcal{F}$ is univalent, one could still use the same construction as in Construction 4.7.*

5 The Rezk Completion

The next aspect in our study of univalent enriched categories, is the *enriched Rezk completion*. There are two features to a suitable Rezk completion for enriched categories. First of all, one needs to show that every enriched category is weakly equivalent to a univalent one (Construction 5.4). This construction is similar to the Rezk completion of categories [2]: in both cases, we construct the enriched Rezk completion as the image of the Yoneda embedding. Note that for Construction 5.4 we assume that V is a complete symmetric monoidal closed category to guarantee that we have an enrichment for the desired presheaf category. Second of all, one needs to prove a universal property (Theorem 5.5). This property says that every enriched functor from an enriched category $\mathcal{E}$ to some univalent enriched category can be extended to the Rezk completion of $\mathcal{E}$.

Our construction of the enriched Rezk completion makes use of the *Yoneda lemma* [12]. As such, we first define representable presheaves and the Yoneda embedding, and we prove the Yoneda lemma.

▶ **Definition 5.1.** *Let V be a complete symmetric monoidal closed category, and let $\mathcal{E}$ be an V-enriched category. Given an object $y : \mathcal{E}$, we define the* ***representable functor*** $\mathsf{r}_0(y) : [\mathcal{E}^{\mathsf{op}}, \mathsf{self}(\mathsf{V})]$ *as follows.*

- *For objects $x : \mathcal{E}$, we define $\mathsf{r}_0(y)\ x \coloneqq \mathcal{E}(x, y)$;*
- *for morphisms $f : x_1 \to x_2$, we define $\mathsf{r}_0(y)\ f$ to be $f^{\mathsf{pre}} : \mathcal{E}(x_2, y) \to \mathcal{E}(x_1, y)$.*

Given a morphism $f : y_1 \to y_2$ in $\mathcal{E}$, we define the ***representable natural transformation*** $\mathsf{r}_1(f) : \mathsf{r}_0(y_1) \to \mathsf{r}_0(y_2)$ *to be $f^{\mathsf{post}} : \mathcal{E}(x, y_1) \to \mathcal{E}(x, y_2)$ for every $x : \mathcal{E}$.*

Finally, the ***enriched Yoneda embedding*** $\mathsf{y}_{\mathcal{E}} : \mathcal{E} \to [\mathcal{E}^{\mathsf{op}}, \mathsf{self}(\mathsf{V})]$ *is defined to be $\mathsf{r}_0(y)$ on objects $y : \mathcal{E}$ and $\mathsf{r}_1(f)$ on morphisms $f : y_1 \to y_2$.*

Note that in Definition 5.1, one also needs to construct V-enrichments for $\mathsf{r}_0(y)$ and $\mathsf{y}_{\mathcal{E}}$, and prove that $\mathsf{r}_1(f)$ is V-enriched. The details for that can be found in [12] and the formalization.

▶ **Proposition 5.2.** *The enriched Yoneda embedding is fully faithful.*

▶ **Problem 5.3.** *Given a category $\mathcal{E}$ enriched over a univalent complete symmetric monoidal category V, to construct a univalent enriched category $\mathcal{R}(\mathcal{E})$ and a weak equivalence $\mathsf{P} : \mathcal{E} \to \mathcal{R}(\mathcal{E})$.*

▶ **Construction 5.4** (for Problem 5.3)**.** We define $\mathcal{R}(\mathcal{E})$ to be the image of the Yoneda embedding $\mathsf{y}_{\mathcal{E}}$. The enriched functor $\mathsf{P} : \mathcal{E} \to \mathcal{R}(\mathcal{E})$ is essentially surjective by construction (Example 4.4). By the Yoneda lemma (Proposition 5.2), P is fully faithful as well. ⌟

Note that Construction 5.4 might increase the universe level. Let us assume that the type of objects of $\mathcal{E}$ and V live in $\mathcal{U}$ and $\mathcal{V}$ respectively. Since objects of $\mathcal{R}(\mathcal{E})$ are enriched presheaves from $\mathcal{E}$ to $\mathsf{self}(\mathsf{V})$ that are in the image of $\mathsf{y}_{\mathcal{E}}$, the type of objects of $\mathcal{R}(\mathcal{E})$ lives in $\mathcal{U} \sqcup \mathcal{V}$. In many examples, $\mathcal{V}$ is a larger universe that $\mathcal{U}$, because we require the category V to have all products indexed by the objects in $\mathcal{E}$.

In addition, we can extend Construction 5.4 to the case where V is not necessarily univalent. To do so, we first take the Rezk completion of monoidal categories [40] of V to obtain a weak equivalence $\mathsf{P} : \mathsf{V} \to \mathcal{R}(\mathsf{V})$. Since P is fully faithful, it preserves underlying categories. Hence, if we have a category $\mathcal{E}$ enriched over V, we obtain a category $\mathsf{P}^*(\mathcal{E})$ enriched over $\mathcal{R}(\mathsf{V})$ using Example 3.7. Then we use Construction 5.4 to obtain the Rezk completion.

Finally, we verify the universal property of the Rezk completion. This property is formulated using the precomposition functor.

▶ **Theorem 5.5.** *Suppose that we have a enriched functor $\mathcal{F} : \mathcal{E}_1 \to \mathcal{E}_2$ and a univalent enriched category $\mathcal{E}_3$. Then the precomposition functor $\mathcal{F}^{\mathsf{pre}} : [\mathcal{E}_2, \mathcal{E}_3] \to [\mathcal{E}_1, \mathcal{E}_3]$ is an adjoint equivalence of categories.*

To verify Theorem 5.5, we use that $\mathcal{E}_3$ is univalent. This implies that the categories $[\mathcal{E}_2, \mathcal{E}_3]$ and $[\mathcal{E}_1, \mathcal{E}_3]$ are both univalent, and thus it suffices to check that $\mathcal{F}^{\mathsf{pre}} : [\mathcal{E}_2, \mathcal{E}_3] \to [\mathcal{E}_1, \mathcal{E}_3]$ is essentially surjective and fully faithful. The proofs of Lemmata 5.6 and 5.7 have some overlap with the ordinary categorical case [2, Theorem 8.4]. However, here we must also check that the obtained functors and natural transformations actually are enriched.

▶ **Lemma 5.6.** *Suppose that we have a enriched functor $\mathcal{F} : \mathcal{E}_1 \to \mathcal{E}_2$ and a univalent enriched category $\mathcal{E}_3$. The functor $\mathcal{F}^{\mathsf{pre}} : [\mathcal{E}_2, \mathcal{E}_3] \to [\mathcal{E}_1, \mathcal{E}_3]$ is fully faithful.*

Proof. The proof that $\mathcal{F}^{\mathsf{pre}} : [\mathcal{E}_2, \mathcal{E}_3] \to [\mathcal{E}_1, \mathcal{E}_3]$ is faithful, is in essence the same as for ordinary categories [2, Lemma 8.1], so we only show that $\mathcal{F}^{\mathsf{pre}}$ is full. Let $\mathcal{G}_1, \mathcal{G}_2 : [\mathcal{E}_2, \mathcal{E}_3]$ be two enriched functors, and suppose that we have an enriched transformation $\tau : \mathcal{F} \cdot \mathcal{G}_1 \Rightarrow \mathcal{F} \cdot \mathcal{G}_2$. We show how to construct the desired enriched natural transformation $\theta : \mathcal{G}_1 \Rightarrow \mathcal{G}_2$.

For all objects $x : \mathcal{E}_2$ the following type is contractible.

$$\sum (f : \mathcal{G}_1\, x \to \mathcal{G}_2\, x), \prod (w : \mathcal{E}_1)(i : \mathcal{F}\, w \cong x), \tau\, w \cdot \mathcal{G}_2\, i = \mathcal{G}_1\, i \cdot f$$

The contractibility of this type follows from our assumption that $\mathcal{F}$ is essentially surjective. From this, we obtain the data of the the desired transformation θ. The fact that θ is V-enriched is shown by using Equation (1) and the fact that $\mathcal{F}$ is essentially surjective. ◀

▶ **Lemma 5.7.** *The functor $\mathcal{F}^{\mathsf{pre}} : [\mathcal{E}_2, \mathcal{E}_3] \to [\mathcal{E}_1, \mathcal{E}_3]$ is essentially surjective.*

Proof. Suppose that we have an enriched functor $\mathcal{G} : [\mathcal{E}_1, \mathcal{E}_3]$. We only demonstrate how to construct the desired enriched functor $\mathcal{H} : [\mathcal{E}_2, \mathcal{E}_3]$.

Suppose that we have $x : \mathcal{E}_2$. Then there is a unique object $y : \mathcal{E}_3$ and function $\varphi : \prod (w : \mathcal{E}_1)(i : \mathcal{F}\, x \cong y), \mathcal{G}\, w \cong y$ such that for all objects $w_1, w_2 : \mathcal{E}_1$, isomorphisms $i_1 : \mathcal{F}\, w_1 \cong x$ and $i_2 : \mathcal{F}\, w_2 \cong x$, and morphisms $k : w_1 \to w_2$ satisfying $\mathcal{F}\, k \cdot i_2 = i_1$, we

have $\mathcal{G}\,k \cdot \varphi\,w_2\,i_2 = \varphi\,w_1\,i_1$. Uniqueness follows from the fact that $\mathcal{F}$ is fully faithful, and the desired element is constructed by using that $\mathcal{F}$ is essentially surjective. One can show that the obtained action on objects gives rise to a functor H from the underlying category of $\mathcal{E}_2$ to that of $\mathcal{E}_3$. We also have isomorphisms $\varphi\,w\,i : \mathcal{G}\,w \cong H\,x$ for all $w : \mathcal{E}_1$ and $i : \mathcal{F}\,x \cong y$.

Next we construct an enrichment for this functor. Suppose, that we have two objects $x, y : \mathcal{E}_2$. Then there is a unique morphism $f : \mathcal{E}_2(x,y) \to \mathcal{E}_3(H\,x, H\,y)$ in V such that for all objects $w_1, w_2 : \mathcal{E}_2$ and isomorphisms $i_1 : \mathcal{F}\,w_1 \cong x$ and $i_2 : \mathcal{F}\,w_2 \cong x$, f is equal to the following composition of morphisms

$$\begin{array}{ccccccc}
\mathcal{E}_2(x,y) & \xrightarrow{i_1^{\mathsf{pre}}} & \mathcal{E}_2(\mathcal{F}\,w_1, y) & \xrightarrow{(i_2^{-1})^{\mathsf{pre}}} & \mathcal{E}_2(\mathcal{F}\,w_1, \mathcal{F}\,w_2) & \xrightarrow{(\mathcal{F})^{-1}} & \mathcal{E}_1(w_1,w_2) \\
 & & & & & \searrow^{\mathcal{G}} & \\
\mathcal{E}_3(\mathcal{G}\,w_1, \mathcal{G}\,w_2) & \xleftarrow[((\varphi\,w_1\,i_1)^{-1})^{\mathsf{pre}}]{} & \mathcal{E}_3(H\,x, \mathcal{G}\,w_2) & \xrightarrow[\varphi\,w_2\,i_2^{\mathsf{post}}]{} & \mathcal{E}_3(H\,x, H\,y) & &
\end{array}$$

This follows from the fact that $\mathcal{F}$ is a weak equivalence. As such, we get the desired enriched functor $\mathcal{H} : [\mathcal{E}_2, \mathcal{E}_3]$. ◀

6 Enriched Monads

We end our study of univalent enriched categories by looking at enriched monads. More specifically, we discuss Kleisli objects (Construction 6.9) in the bicategory of enriched categories. At first glance, it might not seem that univalence plays an interesting role, but upon closer look, this question is rather subtle.

Usually, the Kleisli category of a monad T on a category C is defined to be the category whose objects are objects of C and whose morphisms from x to y are morphisms $x \to \mathsf{T}\,y$ in C. We denote this category by $\mathsf{K}(\mathsf{T})$. In general, this category is not univalent (for example the constant monad on the unit set). This situation can be rectified by defining the Kleisli category in a slightly different way [4], namely as the image of the free algebra functor from C to the Eilenberg-Moore category $\mathsf{EM}(\mathsf{T})$ of T. The resulting univalent category is denoted by $\mathsf{Kleisli}(\mathsf{T})$. To derive the usual theorems about Kleisli categories, one can instantiate the formal theory of monads [13, 28, 36], meaning that it suffices to prove the universal property for Kleisli objects. Proving the desired universal property is a nice exercise using the universal property of the Rezk completion (Theorem 5.5).

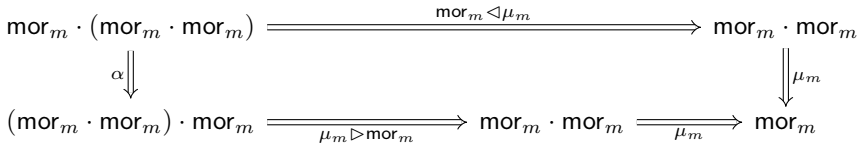
The key notion of this section, enriched monads, can be defined concisely as monads internal to $\mathsf{EnrichCat}_{\mathsf{V}}$. Recall that monads in bicategories are defined as follows.

▶ **Definition 6.1.** *Let* B *be a bicategory. A* ***monad*** m ***in*** B *is given by*

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

such that the following diagrams commute.

$$\begin{array}{ccccccccc}
\mathsf{mor}_m & \xrightarrow{\rho^{-1}} & \mathsf{mor}_m \cdot \mathrm{id}\,x & \xrightarrow{\mathsf{mor}_m \triangleleft \eta_m} & \mathsf{mor}_m \cdot \mathsf{mor}_m & \xleftarrow{\eta_m \triangleright \mathsf{mor}_m} & \mathrm{id} \cdot \mathsf{mor}_m & \xleftarrow{\lambda^{-1}} & \mathsf{mor}_m \\
 & \searrow^{\mathrm{id}} & & & \downarrow{\scriptstyle \mu_m} & & & \swarrow_{\mathrm{id}} & \\
 & & & & \mathsf{mor}_m & & & &
\end{array}$$



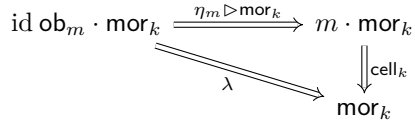
Here λ and ρ are the left and right unitors of B, and α is the associator of B.

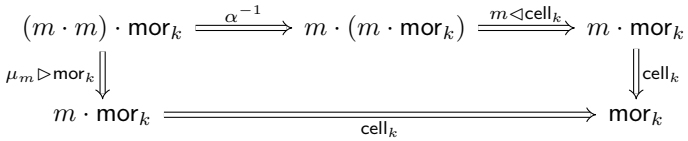
For enriched categories, one can further unfold this definition and phrase it in terms of enrichments. This results in the notion of *enrichments for monads*.

▶ **Definition 6.2.** *Suppose that we have a category C, a monad T on C, and a V-enrichment for C. Then a V-**enrichment** for T consists of a V-enrichment for the endofunctor T such that the unit η_{T} and multiplication μ_{T} are V-enriched natural transformations.*

When we say *enriched monad*, we mean a monad together with an enrichment. In the remainder of this section, we are concerned with *Kleisli objects* in the bicategory of enriched categories. To define Kleisli objects, we first define their cocones. Note that for these definitions, we talk about arbitrary bicategories B and monads internal to B.

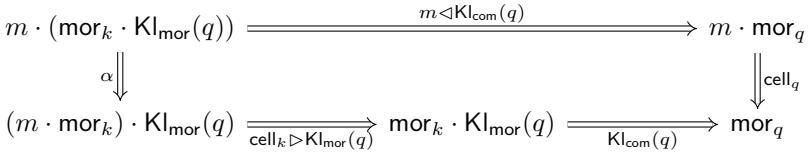
▶ **Definition 6.3.** *Let B be a bicategory and let m be a monad in B. A **Kleisli cocone** k for m in B consists of an object $\mathsf{ob}_k : \mathsf{B}$, a 1-cell $\mathsf{mor}_k : \mathsf{ob}_m \to \mathsf{ob}_k$, and a 2-cell $\mathsf{cell}_k : m \cdot \mathsf{mor}_k \Rightarrow \mathsf{mor}_k$ such that the following diagrams commute.*



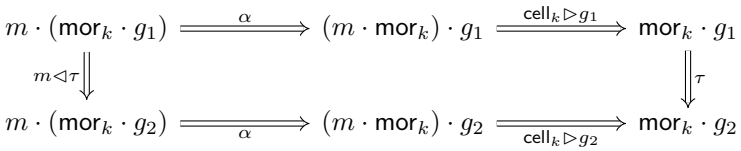


▶ **Definition 6.4.** *A Kleisli cocone k is **universal** if the following conditions are satisfied.*

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



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



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

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

As discussed before, there are multiple ways to define Kleisli categories. We first define an enrichment for $\mathsf{K}(\mathsf{T})$.

▶ **Example 6.5.** *Let* T *be an enriched monad on an enriched category* $\mathcal{E}$*. We define a* V***-enrichment*** $\mathsf{K_e}(\mathsf{T})$ ***for*** $\mathsf{K}(\mathsf{T})$ *as follows.*

- *We define* $\mathsf{K_e}(\mathsf{T})(x, y)$ *to be* $\mathcal{E}(x, \mathsf{T}\, y)$*.*
- *We define* $\mathsf{id^e}(x)$ *to be* $\overrightarrow{\eta_{\mathsf{T}}\, x}$*.*
- *We define* $\mathsf{comp}(x, y, z)$ *as the following composition of morphisms.*

$$\mathcal{E}(y, \mathsf{T}\, z) \otimes \mathcal{E}(x, \mathsf{T}\, y) \xrightarrow{\mathsf{T}\otimes\mathrm{id}} \mathcal{E}(\mathsf{T}\, y, \mathsf{T}\,(\mathsf{T}\, z)) \otimes \mathcal{E}(x, \mathsf{T}\, y) \xrightarrow{\mathsf{comp}} \mathcal{E}(x, \mathsf{T}\,(\mathsf{T}\, z)) \xrightarrow{(\mu_{\mathsf{T}}\, z)^{\mathsf{post}}} \mathcal{E}(x, \mathsf{T}\, z)$$

The operations $\overrightarrow{f}$ *and* $\overleftarrow{f}$ *in* $\mathsf{K_e}(\mathsf{T})$ *are inherited from* $\mathcal{E}$*.*

Next we define an enrichment for $\mathsf{Kleisli}(\mathsf{T})$. Since $\mathsf{Kleisli}(\mathsf{T})$ is defined as a full subcategory of the Eilenberg-Moore category $\mathsf{EM}(\mathsf{T})$, we define an enrichment for $\mathsf{EM}(\mathsf{T})$ first.

▶ **Example 6.6.** *Suppose that* V *has equalizers, and let* T *be an enriched monad on an enriched category* $\mathcal{E}$*. Note that we can define the Eilenberg-Moore category of* T *as a full subcategory of* $\mathsf{Dialg}(\mathsf{T}, \mathrm{id})$*. By Examples 3.2 and 3.4 we obtain the desired* V***-enrichment*** $\mathsf{EM_e}(\mathsf{T})$ ***on*** $\mathsf{EM}(\mathsf{T})$*.*

Using Example 6.6 one can show that $\mathsf{EnrichCat_V}$ has Eilenberg-Moore objects. In general, we have an enriched functor $\mathsf{FreeAlg_T} : \mathcal{E} \to \mathsf{EM_e}(\mathsf{T})$. This functor sends every object x to the free algebra $\mathsf{T}\, x$. Now we define an enrichment for $\mathsf{Kleisli}(\mathsf{T})$.

▶ **Example 6.7.** *Suppose that* V *is a monoidal category with equalizers, and let* T *be an enriched monad on an enriched category* $\mathcal{E}$*. Note that* $\mathsf{Kleisli}(\mathsf{T})$ *is constructed as a full subcategory of the Eilenberg-Moore category, and thus by Example 6.6 we obtain the* V***-enrichment*** $\mathsf{Kleisli_e}(\mathsf{T})$ ***for*** $\mathsf{Kleisli}(\mathsf{T})$*.*

The category defined in Example 6.7 is univalent if we assume $\mathcal{E}$ to be univalent. This is because the Eilenberg-Moore category of a monad on a univalent category is always univalent and because univalence is preserved under full subcategories. In addition, note that in Example 6.7 we assume that V has equalizers, whereas in Example 6.5, we do not.

We finish this section by showing that $\mathsf{Kleisli_e}(\mathsf{T})$ satisfies the required universal property. The main idea behind the proof is that we have a weak equivalence $\mathsf{incl_T} : \mathsf{K_e}(\mathsf{T}) \to \mathsf{Kleisli_e}(\mathsf{T})$, and this weak equivalence allows use to instantiate Theorem 5.5.

▶ **Problem 6.8.** *Given a monoidal category* V *with equalizers, to construct Kleisli objects in the bicategory* $\mathsf{EnrichCat_V}$*.*

▶ **Construction 6.9** (for Problem 6.8)**.** Given an enriched monad T on $\mathcal{E}$, the Kleisli object of T in $\mathsf{EnrichCat_V}$ is given by $\mathsf{Kleisli_e}(\mathsf{T})$. The main work lies in verifying the universal property. This check happens in three steps.

First, we define a weak equivalence $\mathsf{incl_T} : \mathsf{K_e}(\mathsf{T}) \to \mathsf{Kleisli_e}(\mathsf{T})$. This enriched functor sends every object x to the free algebra on x. The action on morphisms is given by the following composition

$$\mathcal{E}(x, \mathsf{T}\, y) \xrightarrow{\mathsf{T}} \mathcal{E}(\mathsf{T}\, x, \mathsf{T}\,(\mathsf{T}\, y)) \xrightarrow{(\mu_y)^{\mathsf{post}}} \mathcal{E}(\mathsf{T}\, x, \mathsf{T}\, y)$$

Second, we check that $\mathsf{K_e}(\mathsf{T})$ gives rise to Kleisli objects in the bicategory of (not necessarily univalent) enriched categories. For this, one can use the same proof as used, for example, by Street [28, Theorem 15].

Third, we conclude that the universal property also holds for $\mathsf{Kleisli}_{\mathsf{e}}(\mathsf{T})$, and we only show how to construct 1-cells arising from the mapping property. Suppose, that we have some Kleisli cocone q in $\mathsf{EnrichedCat}_{\mathsf{V}}$. We get an enriched functor $\mathcal{F} : \mathsf{K}_{\mathsf{e}}(\mathsf{T}) \to \mathsf{ob}_q$. From Theorem 5.5, we get the desired 1-cell $\tilde{\mathcal{F}} : \mathsf{Kleisli}_{\mathsf{e}}(\mathsf{T}) \to \mathsf{ob}_q$. ⌟

Note the similarities between Construction 6.9 and the construction of Kleisli objects for univalent categories [36, Construction 6.10].

7 Conclusion

In this paper, we studied univalent enriched categories, and we discussed several aspects of their study. Our notion of univalent enriched category was based on enrichments, and we viewed enriched categories as a category together with an enrichment. First, we proved a structure identity principle for univalent enriched categories, which we formulated using univalent bicategories. The proof used displayed bicategories. Second, we showed that all weak equivalences between univalent enriched categories are adjoint equivalences. Here we made use of orthogonal factorization systems. Third, we discussed the Rezk completion of enriched categories, which we constructed using the Yoneda lemma. We also used the Rezk completion to construct Kleisli objects in the bicategory of univalent enriched categories.

Along the way, we saw a couple of interesting points where univalence interacted with enrichment. When we defined the change-of-base operation in Example 3.7, we restricted ourselves to lax functors that preserve underlying categories. This was to guarantee that the resulting category would remain univalent. In addition, we assumed that the monoidal category V has equalizers in the construction of the univalent Kleisli category (Example 6.7).

There are several ways to extend the results in this paper. A wide variety of notions in category theory can be defined internally to a bicategory. However, for enriched categories, these internal notions are not always the correct ones. For example, the notion of a fully faithful 1-cell can be defined internally to a bicategory using a representable definition, but the obtained notion does not correspond to the one given in Definition 4.3. To obtain the desired notions, one could use the theory of equipments [39], and one interesting extension of this work would be to develop the equipment of enriched categories. Such work would build forth upon recent work on univalent double (bi)categories [26, 37, 32]. Another interesting extension would be formalizing applications of enriched categories, such as models of the enriched effect calculus [7] or enriched profunctor optics [6].

References

1 Benedikt Ahrens, Dan Frumin, Marco Maggesi, Niccolò Veltri, and Niels van der Weide. Bicategories in univalent foundations. *Mathematical Structures in Computer Science*, pages 1–38, 2022. doi:10.1017/S0960129522000032.

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

3 Benedikt Ahrens, Ralph Matthes, Niels van der Weide, and Kobe Wullaert. Displayed monoidal categories for the semantics of linear logic. In Amin Timany, Dmitriy Traytel, Brigitte Pientka, and Sandrine Blazy, editors, *Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2024, London, UK, January 15-16, 2024*, pages 260–273. ACM, 2024. doi:10.1145/3636501.3636956.

4 Benedikt Ahrens, Paige Randall North, Michael Shulman, and Dimitris Tsementzis. The univalence principle. *CoRR*, abs/2102.06275, 2021. arXiv:2102.06275.

5 Alexander Campbell. Skew-enriched categories. *Appl. Categ. Structures*, 26(3):597–615, 2018. doi:10.1007/s10485-017-9504-0.

6 Bryce Clarke, Derek Elkins, Jeremy Gibbons, Fosco Loregian, Bartosz Milewski, Emily Pillmore, and Mario Román. Profunctor Optics, a Categorical Update. *Compositionality*, 6:1, February 2024. doi:10.32408/compositionality-6-1.

7 Jeff Egger, Rasmus Ejlers Møgelberg, and Alex Simpson. The enriched effect calculus: syntax and semantics. *J. Log. Comput.*, 24(3):615–654, 2014. doi:10.1093/logcom/exs025.

8 David Gepner and Rune Haugseng. Enriched ∞-categories via non-symmetric ∞-operads. *Adv. Math.*, 279:575–716, 2015. doi:10.1016/j.aim.2015.02.007.

9 Paul G. Goerss and John F. Jardine. *Simplicial homotopy theory*. Modern Birkhäuser Classics. Birkhäuser Verlag, Basel, 2009. Reprint of the 1999 edition. doi:10.1007/978-3-0346-0189-4.

10 Jason Z. S. Hu and Jacques Carette. Formalizing category theory in Agda. In Catalin Hritcu and Andrei Popescu, editors, *CPP '21: 10th ACM SIGPLAN International Conference on Certified Programs and Proofs, Virtual Event, Denmark, January 17-19, 2021*, pages 327–342. ACM, 2021. doi:10.1145/3437992.3439922.

11 Krzysztof Kapulkin and Peter LeFanu Lumsdaine. The Simplicial Model of Univalent Foundations (after Voevodsky). *Journal of the European Mathematical Society*, 23(6):2071–2126, 2021. doi:10.4171/JEMS/1050.

12 Max Kelly. *Basic concepts of enriched category theory*, volume 64. CUP Archive, 1982.

13 Stephen Lack and Ross Street. The formal theory of monads. II. *J. Pure Appl. Algebra*, 175(1-3):243–265, 2002. Special volume celebrating the 70th birthday of Professor Max Kelly. doi:10.1016/S0022-4049(02)00137-8.

14 Tom Leinster. A survey of definitions of n-category. *Theory Appl. Categ.*, 10:1–70, 2002.

15 Fosco Loregian and Emily Riehl. Categorical notions of fibration. *Expo. Math.*, 38(4):496–514, 2020. doi:10.1016/j.exmath.2019.02.004.

16 Saunders Mac Lane. *Categories for the working mathematician*, volume 5. Springer Science & Business Media, 2013.

17 Per Martin-Löf. *Intuitionistic type theory*, volume 1 of *Studies in proof theory*. Bibliopolis, 1984.

18 The mathlib Community. The Lean mathematical library. In Jasmin Blanchette and Catalin Hritcu, editors, *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020*, pages 367–381. ACM, 2020. doi:10.1145/3372885.3373824.

19 Dylan McDermott and Alan Mycroft. Galois connecting call-by-value and call-by-name. In Amy P. Felty, editor, *7th International Conference on Formal Structures for Computation and Deduction, FSCD 2022, August 2-5, 2022, Haifa, Israel*, volume 228 of *LIPIcs*, pages 32:1–32:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.FSCD.2022.32.

20 Dylan McDermott and Tarmo Uustalu. What makes a strong monad? In Jeremy Gibbons and Max S. New, editors, *Proceedings Ninth Workshop on Mathematically Structured Functional Programming, MSFP@ETAPS 2022, Munich, Germany, 2nd April 2022*, volume 360 of *EPTCS*, pages 113–133, 2022. doi:10.4204/EPTCS.360.6.

21 Eugenio Moggi. Computational lambda-calculus and monads. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89), Pacific Grove, California, USA, June 5-8, 1989*, pages 14–23. IEEE Computer Society, 1989. doi:10.1109/LICS.1989.39155.

22 Ulf Norell. Dependently typed programming in Agda. In *Proceedings of the 4th international workshop on Types in language design and implementation*, pages 1–2, 2009.

23 Gordon Plotkin and John Power. Notions of computation determine monads. In *International Conference on Foundations of Software Science and Computation Structures*, pages 342–356. Springer, 2002.

24 Gordon Plotkin and John Power. Algebraic operations and generic effects. *Applied categorical structures*, 11(1):69–94, 2003.

25 John Power. Models for the computational lambda-calculus. In Ted Hurley, Mícheál Mac an Airchinnigh, Michel P. Schellekens, and Anthony Karel Seda, editors, *First Irish Conference on the Mathematical Foundations of Computer Science and Information Technology, MFCSIT 2000, Cork, Ireland, July 20-21, 2000*, volume 40 of *Electronic Notes in Theoretical Computer Science*, pages 288–301. Elsevier, 2000. doi:10.1016/S1571-0661(05)80056-8.

26 Nima Rasekh, Niels van der Weide, Benedikt Ahrens, and Paige Randall North. Insights From Univalent Foundations: A Case Study Using Double Categories. *CoRR*, abs/2402.05265, 2024. doi:10.48550/arXiv.2402.05265.

27 Egbert Rijke. Introduction to homotopy type theory. *arXiv preprint arXiv:2212.11082*, 2022.

28 Ross Street. The formal theory of monads. *J. Pure Appl. Algebra*, 2(2):149–168, 1972. doi:10.1016/0022-4049(72)90019-9.

29 The Coq Development Team. The Coq Proof Assistant, September 2022. doi:10.5281/zenodo.7313584.

30 Christopher Francis Townsend. *Preframe techniques in constructive locale theory*. PhD thesis, Imperial College London (University of London), 1996.

31 The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. https://homotopytypetheory.org/book, Institute for Advanced Study, 2013.

32 Dominic Verity. Enriched categories, internal categories and change of base. *Repr. Theory Appl. Categ.*, 20:1–266, 2011.

33 Vladimir Voevodsky, Benedikt Ahrens, Daniel Grayson, et al. Unimath — a computer-checked library of univalent mathematics. available at http://unimath.org. doi:10.5281/zenodo.10849216.

34 Mitchell Wand. Fixed-point constructions in order-enriched categories. *Theor. Comput. Sci.*, 8:13–30, 1979. doi:10.1016/0304-3975(79)90053-7.

35 Charles A. Weibel. *An introduction to homological algebra*, volume 38 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Cambridge, 1994. doi:10.1017/CBO9781139644136.

36 Niels van der Weide. The formal theory of monads, univalently. In Marco Gaboardi and Femke van Raamsdonk, editors, *8th International Conference on Formal Structures for Computation and Deduction, FSCD 2023, July 3-6, 2023, Rome, Italy*, volume 260 of *LIPIcs*, pages 6:1–6:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.FSCD.2023.6.

37 Niels van der Weide, Nima Rasekh, Benedikt Ahrens, and Paige Randall North. Univalent double categories. In Amin Timany, Dmitriy Traytel, Brigitte Pientka, and Sandrine Blazy, editors, *Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2024, London, UK, January 15-16, 2024*, pages 246–259. ACM, 2024. doi:10.1145/3636501.3636955.

38 John Wiegley. category-theory: Category Theory in Coq, 2023. URL: https://github.com/jwiegley/category-theory.

39 R. J. Wood. Abstract pro arrows I. *Cahiers de topologie et géométrie différentielle*, 23(3):279–290, 1982.

40 Kobe Wullaert, Ralph Matthes, and Benedikt Ahrens. Univalent monoidal categories. In Delia Kesner and Pierre-Marie Pédrot, editors, *28th International Conference on Types for Proofs and Programs, TYPES 2022, June 20-25, 2022, LS2N, University of Nantes, France*, volume 269 of *LIPIcs*, pages 15:1–15:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.TYPES.2022.15.

The Flower Calculus

Pablo Donato
LIX, École Polytechnique, Palaiseau, France

Abstract

We introduce the flower calculus, a deep inference proof system for intuitionistic first-order logic inspired by Peirce's existential graphs. It works as a rewriting system over inductive objects called "flowers", that enjoy both a graphical interpretation as topological diagrams, and a textual presentation as nested sequents akin to coherent formulas. Importantly, the calculus dispenses completely with the traditional notion of symbolic connective, operating solely on nested flowers containing atomic predicates. We prove both the soundness of the full calculus and the completeness of an analytic fragment with respect to Kripke semantics. This provides to our knowledge the first analyticity result for a proof system based on existential graphs, adapting semantic cut-elimination techniques to a deep inference setting. Furthermore, the kernel of rules targetted by completeness is fully invertible, a desirable property for both automated and interactive proof search.

2012 ACM Subject Classification Theory of computation → Proof theory; Theory of computation → Constructive mathematics

Keywords and phrases deep inference, graphical calculi, existential graphs, intuitionistic logic, Kripke semantics, cut-elimination

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.5

Related Version *Full Version*: `https://arxiv.org/abs/2402.15174` [21]

Supplementary Material
Software (Mechanized Theory): `https://github.com/Champitoad/flowers-metatheory` [19]
archived at `swh:1:dir:290076a847ca95e93c17fb66659086d7f68be014`
Software (Online Demo): `https://github.com/Champitoad/flower-prover` [20]
archived at `swh:1:dir:fcc934cae3a75692c031dc82ffdab138084a472d`

Acknowledgements I want to thank Luc Chabassier for writing the Lua script that was used to generate all the flower drawings in this document, and Benjamin Werner for useful feedback on a first draft of this paper. Lastly, I thank Tito for teaching me some invaluable formatting tricks.

1 Introduction

Graphical proof building. Proof assistants – also called *interactive theorem provers* (ITPs) – provide a set of tools to ease the process of formalizing mathematical developments. This includes languages to specify definitions and statements conveniently, but also interfaces to build proofs interactively without having to fill in all the details. The dominant paradigm for these interfaces is that of *tactic languages* [44]: the user is exposed with a set of *goals* that remain to be proved, constituting the *proof state*, and modifies these goals through textual commands, called *tactics*, until there is no goal left. This is currently what is implemented in mainstream proof assistants such as Coq [58] and Lean [45].

In recent years, there have been several efforts to replace or complement textual tactic languages with *graphical user interfaces* (GUIs) [51, 4, 38, 12, 53, 35, 68, 3]. The hope is to make proof assistants more intuitive and accessible to beginners and non-specialists, but also, to some extent, more productive and ergonomic even for experts.

The initial motivation for this work was to design a proof calculus well-suited to *direct manipulation* in such a graphical setting. The idea is that the user should be able to interact directly with the graphical representation of the proof state, using a pointing device such as

a mouse or fingers on a touch screen. In previous work [22], we proposed a way to synthesize complex logical inferences through *drag-and-drop* actions between two formulas of the current goal/sequent, based on the *subformula linking* (SFL) methodology [12, 13].

Diagrammatic reasoning. In this work, we show that (single-conclusion) sequents and symbolic formulas built from binary connectives and unary quantifiers are not mandatory for representing the proof state. Other authors have defended the idea of using *diagrams* as a more user-friendly frontend for ITPs. In particular, Linker et al. showed how to integrate tactic-based automation in an ITP based on *spider diagrams* [35], which are equivalent in expressive power to classical monadic *first-order logic* (FOL) [29].

We introduce a new data structure for goals inspired by an earlier invention in the history of diagrammatic logic: the *existential graphs* (EGs) of C. S. Peirce [52]. We noticed that our structure could be drawn and manipulated metaphorically in the form of nested *flowers*, and thus chose to name *flower calculus* the proof system for full intuitionistic FOL that we built around it. Our focus in this paper will be to introduce the flower calculus to readers unfamiliar with EGs, and to study its fundamental properties through the lens of modern *structural proof theory*.

Implementation. We have formalized in Coq a bidirectional simulation between the flower calculus and cut-free sequent calculus, yielding a soundness theorem and a *weak* completeness theorem for an analytic fragment of the flower calculus [19]. In this paper, we follow a *semantic* rather than syntactic approach, avoiding translations to and from symbolic formulas to obtain a stronger completeness result.

While currently at an early stage, we are also developing the *Flower Prover*, a prototype of direct-manipulation GUI for ITPs based on the flower calculus [20]. The interested reader can try a publicly available version of the prototype online[1]. We leave a detailed account of the Flower Prover and its connection to the flower calculus for future work.

Outline. The article is organized as follows: in Section 2 we give a brief overview of the original diagrammatic syntax of EGs used by Peirce in his system Alpha for classical propositional logic. In Section 3 we retrace the origin of an intuitionistic variant of EGs first introduced by Oostra in [46], that directly inspired our flower metaphor. In Section 4 we illustrate quickly the original mechanism of lines of identity used by Peirce to express first-order quantifiers in his Beta system, and show how to recast it in a more traditional binder-based syntax. In Section 5 we introduce our inductive syntax for flowers, and in Section 6 we give the full set of inference rules of the flower calculus as well as our notion of proof. In Section 7 we give a direct Kripke semantics to flowers, and in Section 8 we show that a restricted fragment of analytic and invertible rules is complete with respect to the semantics. Finally we conclude in Section 9 by a comparison with some related works.

▶ Note. The full version of this paper with complete appendices is available on arXiv [21]. The proof of soundness of the flower calculus is given in [21, Appendix B]. Contrary to the completeness proof, it is mostly routine work that does not require much insight. Detailed proofs for the deduction and completeness theorems are given respectively in [21, Appendix C.1] and [21, Appendix C.2]. Readers already familiar with EGs can find a detailed comparison of the rules of the flower calculus with Peirce's illative transformations in [21, Appendix A].

[1] `https://www.lix.polytechnique.fr/Labo/Pablo.DONATO/flowerprover/`

2 Existential graphs

Peirce designed in total three systems of EGs, which he called respectively Alpha, Beta and Gamma. They were invented chronologically in that order, which also captures their relationship in terms of complexity: Alpha is the foundation on which the other systems are built, and can today be understood as a diagrammatic calculus for classical *propositional* logic. As we will see in Section 4, Beta corresponds to a variable-free representation of *first-order* logic without function symbols. The last system Gamma is more experimental, with various unfinished features that have been interpreted as attempts to capture *modal* [67] and *higher-order* logics.

Sheet of Assertions. The most fundamental concept of Alpha is the *sheet of assertion*, denoted by SA thereafter. It is the space where statements are scribed by the reasoner, typically a sheet of paper, a blackboard, or a computer display. As its name indicates, scribing a statement on SA amounts to *asserting its truth*. Thus naturally, the empty SA where nothing is scribed will denote *vacuous truth*, traditionally signified by the symbol $\top$.

Juxtaposition. As we know from natural deduction, asserting the truth of the *conjunction* $a \wedge b$ of two propositions a and b, amounts to asserting *both* the truth of a and the truth of b. In Alpha, there is no need to introduce the symbolic connective $\wedge$, since one can just write both a and b at distinct locations on SA:

$a \quad b$

More generally, one might consider any two portions G and H of SA, and interpret their *juxtaposition* $G\ H$ as signifying that we assert the truth of their conjunction.

Cuts. Asserting the truth of the *negation* $\neg a$ of a proposition a, amounts to *denying* the truth of a. This is done in Alpha by *enclosing* a in a closed curve like so:

Peirce called such curves *cuts*[2], because they ought to be seen as literal cuts in the paper sheet that embodies SA. Note that they do not need to be circles: all that matters is that a is in a separate area from the rest of SA. This is precisely the content of the *Jordan curve theorem* in topology, and thus we can take cuts to be arbitrary Jordan curves. This entails in particular that cuts cannot intersect each other, but can be freely nested. Then as for juxtaposition, one can replace the proposition a in the interior of the cut by any *graph* G – i.e. any portion of SA – as long as the cut does not intersect other cuts in G.

Relationship with formulas. With just these two *icons*, juxtaposition and cuts, one can therefore assert the truth of any proposition made up of conjunctions and negations and built from atomic propositions. Importantly, the only symbols needed for doing so are letters $a, b, c \ldots$ denoting atomic propositions, that is "pure" symbols that do not have any logical meaning associated to them.

[2] Not to be confused with the name given to instances of the *cut rule* in sequent calculus.

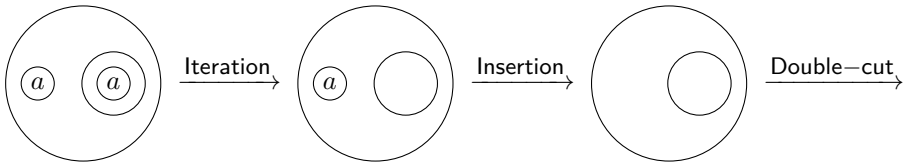
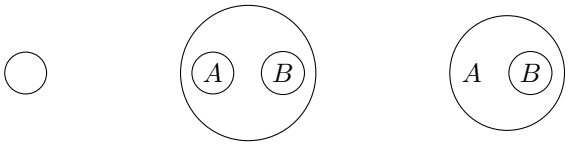


Figure 1 Proof of the law of excluded middle in Alpha.

Now, it is well-known that $\{\wedge, \neg\}$ is *functionally complete*, meaning that any boolean truth function can be expressed as the composition of conjunctions and negations. In particular, the symbolic definitions of *falsehood* $\bot \triangleq \neg\top$, classical *disjunction* $A \vee B \triangleq \neg(\neg A \wedge \neg B)$ and classical *implication* $A \supset B \triangleq \neg(A \wedge \neg B)$ can be expressed by the following three graphs:

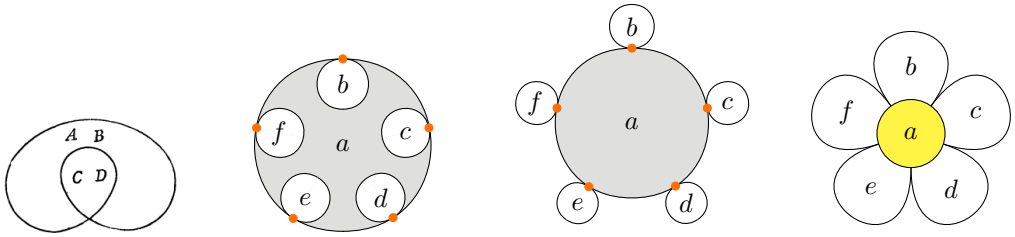


Thus one can easily encode any propositional formula into a classically equivalent graph. Conversely, one can translate any graph into a classically equivalent formula, as has been shown for instance in [54]. In fact, there are usually many possible formula readings of a given graph. One reason is that juxtaposition of graphs is a *variadic* operation, as opposed to conjunction of formulas which is *dyadic*: thus formulas that only differ up to *associativity* are associated to the same graph. Also, thanks to the topological nature of SA, juxtaposition is naturally *commutative*: the locations of two juxtaposed graphs do not matter, as long as they live in the same area delimited by a cut. The combination of these properties is called the *isotropy* of SA in [40], and is captured in traditional proof theory through the use of *(multi)sets* for modelling contexts in sequents.

Illative transformations. In order to have a proof system, one needs a collection of *inference rules* for deducing true statements from other true statements. In Alpha, inference rules are implemented by what Peirce called *illative transformations* on graphs. In modern terminology, they correspond to *rewriting* rules that can be applied to any subgraph. By measuring the depth of a subgraph as the number of cuts in which it is enclosed, we thus have that the rules of Alpha are applicable on subgraphs of arbitrary depth. This makes Alpha deserving of the title of *deep inference* system.

Figure 1 shows a proof of the law of excluded middle $a \vee \neg a$ in Alpha. The first step consists in applying the illative transformation of Iteration to erase the subgraph $\textcircled{a}$. More generally, Iteration allows to erase any subgraph G as long as G already occurs "higher" in SA, i.e. in an area that encloses the erased occurrence of G. The second step of Insertion allows to erase the other occurrence of $\textcircled{a}$ because it is scribed in a *negative* area, i.e. an area enclosed in an *odd* number of cuts – 1 in this case[3]. The last step of Double−cut allows to *collapse* the two remaining cuts, because there is nothing but empty space in between them. This leaves us with the empty SA, having thus reduced the initial goal to trivial truth.

[3] It might be quite confusing that we call "Insertion" a transformation that *erases* information. This is because we use Peirce's original terminology, despite the fact that we adopt a *backward* reading of rules where the conclusion that we want to prove is reduced to a sufficient premiss.



(a) Peirce's scroll. **(b)** Oostra's curl. **(c)** Inside-out curl. **(d)** Flower.

Figure 2 From scrolls to flowers.

3 Flowers

The scroll. In [50, pp. 533–535], Peirce explains that he did not immediately come up with the idea of juxtaposition and cuts as diagrammatizations of conjunction and negation. Instead, they arose as the natural development of a more primitive icon that he called the *scroll*. Figure 2a shows Peirce's drawing of the scroll as it appears in [50, Fig. 5]. He defines its intended meaning as that of a "conditional de inesse", which corresponds to the material implication of classical logic. Then the graph of Figure 2a is interpreted as the formula $(A \wedge B) \supset (C \wedge D)$. This agrees with the encoding of implication given in Section 2, if one sees the outer boundary enclosing the antecedent $A\ B$ and the inner boundary enclosing the consequent $C\ D$ as nested cuts.

It is no coincidence that Peirce based his most fundamental icon on implication: according to Lewis [34, p. 79], he was the one who introduced the "illative relation" of implication into symbolic logic in the first place, by giving it a distinguished symbol and studying extensively the algebraic laws that govern it (e.g. Peirce's law $((A \supset B) \supset A) \supset A$).

The n-ary scroll. In order to interpret the scroll as an *intuitionistic* implication, Oostra proposed in [46] to reify the scroll as a primitive icon of EGs, distinguished from the nesting of two cuts. In fact he went further, by generalizing both the cut and the scroll into an n-ary construction called the *curl*, where n is the number of inner boundaries, called *loops*. Figure 2b shows an example of curl with five loops, where the unique intersection points between inner and outer boundaries are highlighted in *orange*[4]. In [40], the curl is simply called *n-ary scroll*, the outer boundary *outloop*, and the inner boundaries *inloops*. Then cuts and scrolls are indeed special cases of n-ary scrolls, respectively with $n = 0$ and $n = 1$.

Like the unary scroll, the n-ary scroll is to be read as an implication whose antecedent is the content of the outloop, and consequent the content of the inloops. The generalization consists in taking the *disjunction* of the contents of all inloops: this reflects nicely the etymological meaning of the word "disjunction", since the inloops enclose *disjoint* areas of the outloop to which they are attached. Then the 5-ary scroll of Figure 2b can be read as the formula $a \supset (b \vee c \vee d \vee e \vee f)$; and the 0-ary scroll obtained by removing all inloops from the latter as $a \supset \bot$, since a 0-ary disjunction is naturally evaluated to its neutral element $\bot$. This coincides with the intuitionistic reading of negation $\neg A \triangleq A \supset \bot$.

Continuity. With this interpretation of the n-ary scroll, the Alpha encodings of disjunction and implication as nested cuts given in Section 2 are no longer valid, because they are not intuitionistically equivalent to the associated binary and unary scrolls. This is illustrated in

[4] We also shade the negative area delimited by the outer boundary in *gray*.

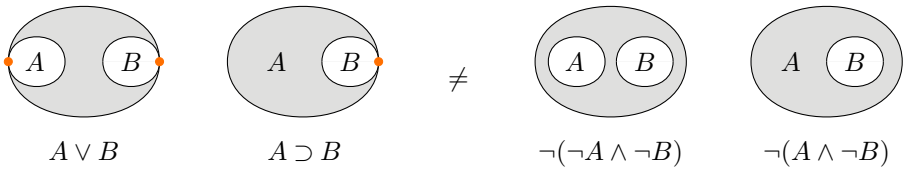


Figure 3 Continuity, disjunction and implication in intuitionistic EGs.

Figure 3, where the closeness in meaning is reflected iconically (but not symbolically) in the fact that the graphs only differ in the *continuity* (or lack thereof) between inloops and their outloop.

▶ Remark 3.1. This might be related to other manifestations of the notion of continuity in the semantics of intuitionistic logic, such as the well-known Stone-Tarski interpretation of formulas as topological spaces [57], and the interpretation of proofs as continuous maps in the *denotational semantics* of Dana Scott[5] [1].

Blooming. In terms of ergonomy, the n-ary scroll has one notable flaw, also shared with the classical cut-based syntax: it quickly induces heavy nestings of curves in the plane, making even relatively simple graphs hard to read for an untrained eye. Our solution is to turn inloops *inside-out*, as illustrated in Figure 2c. In this way, we effectively divide the amount of curve-nesting in scrolls by two. And as an added bonus, the new icon is reminiscent of a *flower*, as if it had bloomed from its curled bud; or as if the pistol cylinder from Figure 2b had transformed into a *pistil*, and its bullet chambers into *petals*.

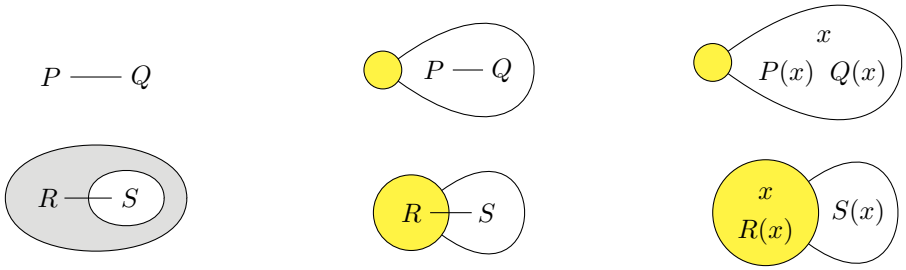
From that point onwards, we chose to fully embrace the flower metaphor: first in our drawing style as witnessed in Figure 2d, but also in our syntactic terminology, to be introduced in the next pages. Negative (resp. positive) outloops are now drawn as *yellow* (resp. *white*) pistils for a slightly more colorful experience, and inloops as transparent petals, i.e. of the same color as the area on which they are scribed.

4 Gardens

Lines of identity. To express first-order quantification, Peirce introduced in Beta the icon of *lines of identity* (LoIs). In short, the usual binders and variables of predicate calculus are replaced by *lines* that connect the occurrences of bound variables in predicate arguments to their binding point. For instance, the formulas $\exists x.P(x) \wedge Q(x)$ and $\forall x.R(x) \supset S(x)$ can be represented in Beta by the graphs of Figure 4a.

The kind of quantification is determined by the location of the binding point, which is taken to be the *outermost* point in the line: if it is in a *positive* area as in the upper graph, then the quantifier is *existential*; otherwise if it is in a *negative* area as in the lower graph, the quantifier is *universal*. This is justified by De Morgan's laws: the lower graph can also be read as the classically equivalent formula $\neg\, \exists x.R(x) \wedge \neg\, S(x)$.

[5] Before the advent of Oostra's intuitionistic EGs, Zalamea gave a detailed analysis of Peirce's philosophy of the *continuum*, how it relates to modern developments in mathematics, and how it is embodied in EGs [66]. Actually according to Oostra [49, p. 162], "the possibility of developing intuitionistic existential graphs was first suggested by Zalamea in the 1990s [64, 65]".



(a) EGs with LoIs. **(b)** Flowers with LoIs. **(c)** Flowers with variables.

Figure 4 From LoIs to variables.

Intuitionistic quantification. In intuitionistic logic however, De Morgan's laws do not hold anymore. Thus in the flower calculus we need a different way to interpret LoIs as quantifiers. Our key insight is to adopt a *polarity-invariant* viewpoint: a LoI now has *existential* (resp. *universal*) force when its outermost point is located in a *petal* (resp. *pistil*). In particular, this implies that LoIs cannot occur at the top-level of SA anymore, but only inside flowers. Thus the two previous Beta graphs are transformed into the single-petal flowers of Figure 4b.

Variables. Quine experimented with a notation similar to LoIs, but deemed it "too cumbersome for practical use" [52, p. 125]. While his lines connected locations inside symbolic formulas written in linear notation, it is true that having a line for each occurrence of bound variable can quickly lead to unreadable diagrams ridden with overlapping lines. This is not a problem in the context of Peirce's work, because his aim was "to separate [relational] reasoning into its smallest steps, [...] not to facilitate reasoning, but to facilitate the study of reasoning" [52, p. 111]; and recent formalizations of the algebra of LoIs in category theory support the pertinence of Peirce's approach [27, 6].

However, keeping in mind our goal of laying the basis for a calculus well-suited to practical reasoning in ITPs, we chose to replace LoIs by a more traditional syntax based on binders and variables. The idea is to substitute every LoI with a variable *binder* scribed in the area of its outermost point, so that the two flowers of Figure 4b transform into those of Figure 4c. Areas delimited by pistils and petals now comprise both flowers and binders, which can be seen metaphorically as *sprinklers* that irrigate the leaves (atomic predicates) of flowers through invisible LoIs, imagined as underground hoses. Hence we call these areas *gardens*.

5 Syntax

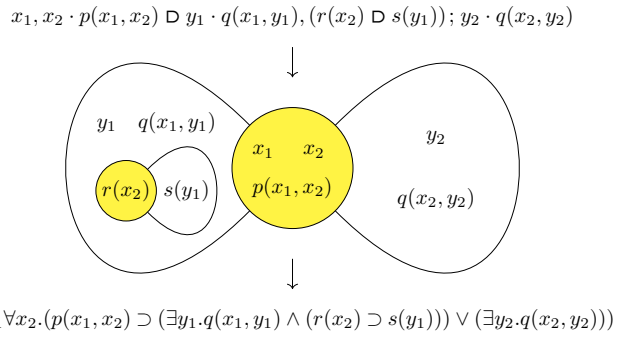
We are now going to distill the syntactic essence of flowers into an inductive, (multi)set-based data structure. This will allow for a more compact textual notation, that is better suited to proof-theoretical study. We previously illustrated how flowers allow to represent purely relational statements without function symbols. Since functions are just deterministic relations, one can in principle formalize any first-order theory in this syntax[6].

▶ **Definition 5.1.** *A* first-order signature *is a pair* $\Sigma = (\mathcal{P}, \mathsf{ar})$*, where* $\mathcal{P}$ *is the countable set of* predicate symbols *of* Σ*, and* $\mathsf{ar} : \mathcal{P} \to \mathbb{N}$ *gives an* arity *to each symbol.*

[6] Conversely, every relation can be faithfully encoded as its characteristic function, which is the basis for the formalization of mathematics in *type theories*.

Kind	Letters
Variables ($\mathcal{V}$)	x, y, z
Flowers ($\mathbb{F}$)	ϕ, ψ, ξ
Gardens ($\mathbb{G}$)	γ, δ
Sprinklers	$\mathbf{x}, \mathbf{y}, \mathbf{z}$
Variable vectors	$\vec{\mathbf{x}}, \vec{\mathbf{y}}, \vec{\mathbf{z}}$
Substitutions	σ, τ
Bouquets	Φ, Ψ, Ξ
Corollas	Γ, Δ
Contexts	$\hat{\Phi}, \hat{\Psi}, \hat{\Xi}$
Theories	$\mathcal{T}, \mathcal{U}$

(a) Conventions for meta-variables.



(b) Interpreting flowers.

Figure 5 Notations.

In the following, we fix a countable set of variables $\mathcal{V}$ and a first-order signature Σ.

▶ **Definition 5.2.** *The sets of* flowers $\mathbb{F}$ *and* gardens $\mathbb{G}$ *are defined by mutual induction:*

Atom *If* $p \in \mathcal{P}$ *and* $\vec{\mathbf{x}} \in \mathcal{V}^{\mathsf{ar}(p)}$*, then* $p(\vec{\mathbf{x}}) \in \mathbb{F}$*;*

Garden *If* $\mathbf{x} \subset \mathcal{V}$ *is a finite set and* $\Phi \subset \mathbb{F}$ *a finite multiset, then* $\mathbf{x} \cdot \Phi \in \mathbb{G}$*;*

Flower *If* $\gamma \in \mathbb{G}$ *and* $\Delta \subset \mathbb{G}$ *is a finite multiset, then* $\gamma \Rsh \Delta \in \mathbb{F}$*.*

Similarly to nested sequents*, the syntax of flowers* ϕ, ψ *and gardens* γ, δ *can be expressed succinctly with the following grammar:*

$$\phi, \psi ::= p(x_1, \ldots, x_n) \mid \gamma \Rsh \delta_1 ; \ldots ; \delta_n \qquad \gamma, \delta ::= x_1, \ldots, x_n \cdot \phi_1, \ldots, \phi_n$$

Building on our botanical metaphor, any finite set $\mathbf{x} \subset \mathcal{V}$ of variables is called a *sprinkler*, finite multiset $\Phi \subset \mathbb{F}$ of flowers a *bouquet*, and finite multiset $\Gamma \subset \mathbb{G}$ of gardens a *corolla*. Following the grammar presentation, we will often write gardens as $x_1, \ldots, x_n \cdot \phi_1, \ldots, \phi_m$, where the x_i are called *binders*; and non-atomic flowers as $\gamma \Rsh \delta_1 ; \ldots ; \delta_n$, where γ is the *pistil* and the δ_i are the *petals*. We write $\{E_i\}_i^n$ to denote a finite (multi)set of size n with elements E_i indexed by $1 \leq i \leq n$. We also omit writing the empty (multi)set, accounting for it with blank space as is done in sequent notation; in particular, $\cdot$ stands for the empty garden $\varnothing \cdot \varnothing$, $\gamma \Rsh$ for the flower with no petals $\gamma \Rsh \varnothing$, and $\gamma \Rsh \cdot$ for the flower with one empty petal.

Note that the order of precedence of operators is $, < \cdot < ; < \Rsh$: this is illustrated in Figure 5b, where a flower expression is parsed into the corresponding flower drawing, and then translated as a formula. Also to improve readability, we will most of the time omit the garden dot "$\cdot$" when the sprinkler is empty, writing Φ instead of $\cdot \Phi$.

▶ Remark 5.3. In some places the choice of letter for meta-variables will be important to disambiguate the kind of syntactic object we denote. Table 5a summarizes our chosen notational conventions in this respect.

We now proceed with routine definitions for handling variables.

▶ **Definition 5.4.** *The sets of* free variables $\mathsf{fv}(-)$ *and* bound variables $\mathsf{bv}(-)$ *of a flower/bouquet/garden are defined recursively by:*

$$\mathsf{fv}(p(\vec{\mathbf{x}})) = \vec{\mathbf{x}} \qquad \mathsf{fv}(\Phi) = \bigcup_{\phi\in\Phi} \mathsf{fv}(\phi) \qquad \mathsf{fv}(\mathbf{x}\cdot\Phi) = \mathsf{fv}(\Phi)\setminus\mathbf{x}$$

$$\mathsf{fv}(\mathbf{x}\cdot\Phi \rhd \Delta) = \mathsf{fv}(\mathbf{x}\cdot\Phi) \cup \bigcup_{\mathbf{y}\cdot\Psi\in\Delta} \mathsf{fv}(\mathbf{x},\mathbf{y}\cdot\Psi)$$

$$\mathsf{bv}(p(\vec{\mathbf{x}})) = \varnothing \quad \mathsf{bv}(\Phi) = \bigcup_{\phi\in\Phi} \mathsf{bv}(\phi) \quad \mathsf{bv}(\mathbf{x}\cdot\Phi) = \mathbf{x}\cup\mathsf{bv}(\Phi) \quad \mathsf{bv}(\gamma \rhd \Delta) = \mathsf{bv}(\gamma)\cup\bigcup_{\delta\in\Delta} \mathsf{bv}(\delta)$$

To avoid reasoning about α-equivalence, we adopt in this work the so-called *Barendregt convention* that all variable binders are distinct, both among themselves and from free variables. Formally, we assume that for any bouquet Φ the two following conditions hold:

1. computing $\mathsf{bv}(\Phi)$ as a multiset gives the same result as computing it as a set;
2. $\mathsf{bv}(\Phi)\cap\mathsf{fv}(\Phi) = \varnothing$.

To define substitutions, we introduce a general notion of *function update*, which will be useful for the semantic evaluation of flowers in Section 7.

▶ **Definition 5.5.** *Let A, B be two sets, $f, g : A \to B$ two functions and $R \subseteq A$ some subset of their domain. The* update *of f on R with g is the function defined by:*

$$(f \mid_R g)(x) = \begin{cases} g(x) & \text{if } x \in R \\ f(x) & \text{otherwise} \end{cases}$$

$- \mid_- -$ is left-associative, that is $f \mid_R g \mid_S h = (f \mid_R g) \mid_S h$. Also if f or g is the identity function $\mathbf{1}$ we omit writing it, i.e. $f \mid_R = f \mid_R \mathbf{1}$ and $\mid_R g = \mathbf{1} \mid_R g$.

▶ **Definition 5.6.** *A* substitution *is a function $\sigma : \mathcal{V} \to \mathcal{V}$ with a finite* support $\mathsf{supp}(\sigma) = \{x \mid \sigma(x) \neq x\}$. *We write $\sigma : \mathbf{x}$ to denote a substitution σ whose support is $\mathbf{x}$. The domain of substitutions is extended to flowers, bouquets and gardens mutually recursively by:*

$$\sigma(p(x_1,\dots,x_n)) = p(\sigma(x_1),\dots,\sigma(x_n)) \qquad \sigma(\phi_1,\dots,\phi_n) = \sigma(\phi_1),\dots,\sigma(\phi_n)$$

$$\sigma(\mathbf{x}\cdot\Phi) = \mathbf{x}\cdot\sigma|_{\mathbf{x}}(\Phi) \qquad \sigma(\mathbf{x}\cdot\Phi \rhd \delta_1 ; \dots ; \delta_n) = \sigma(\mathbf{x}\cdot\Phi) \rhd \sigma|_{\mathbf{x}}(\delta_1) ; \dots ; \sigma|_{\mathbf{x}}(\delta_n)$$

We say that a substitution $\sigma : \mathbf{x}$ is capture-avoiding *in a bouquet Φ if $\sigma(\mathbf{x})\cap\mathsf{bv}(\Phi) = \varnothing$.*

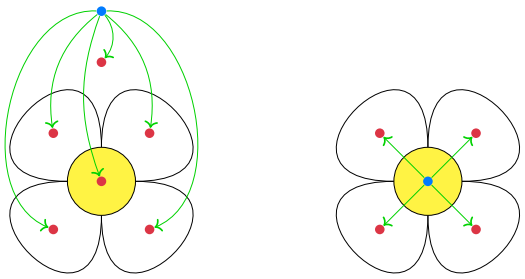
6 Calculus

Equipped with an inductive syntax, we can now express formally the inference rules of the flower calculus. First we need a notion of *context* to apply rules at arbitrarily deep locations:

▶ **Definition 6.1** (Context)**.** Contexts $\hat{\Phi}$ *are defined inductively by the following grammar:*

$$\hat{\Phi}, \hat{\Psi}, \hat{\Xi} ::= \Psi, \hat{\phi} \qquad \hat{\phi}, \hat{\psi}, \hat{\xi} ::= \square \mid \mathbf{x}\cdot\hat{\Phi} \rhd \Delta \mid \gamma \rhd \mathbf{x}\cdot\hat{\Phi} ; \Delta$$

Informally, a context can be seen as a bouquet with exactly one occurrence of a special flower $\square$ called its hole. *The* filling *of a context $\hat{\Phi}$ with a bouquet Ψ (resp. context $\hat{\Psi}$) is the bouquet $\hat{\Phi}\{\Psi\}$ (resp. context $\hat{\Phi}\{\hat{\Psi}\}$) equal to $\hat{\Phi}$ where $\square$ has been substituted with Ψ (resp. $\hat{\Psi}$).*



(a) Cross-pollination. **(b)** Self-pollination.

Figure 6 Pollination in flowers.

▶ **Definition 6.2** (Polarity). *The number of* inversions $\mathsf{inv}(\hat{\Phi})$ *of a context* $\hat{\Phi}$ *is:*

$$\mathsf{inv}(\Box) = 0 \quad \mathsf{inv}(\Psi, \hat{\phi}) = \mathsf{inv}(\hat{\phi}) \quad \mathsf{inv}(\mathbf{x} \cdot \hat{\Phi} \rhd \Delta) = 1 + \mathsf{inv}(\hat{\Phi}) \quad \mathsf{inv}(\gamma \rhd \mathbf{x} \cdot \hat{\Phi}; \Delta) = \mathsf{inv}(\hat{\Phi})$$

We say that a context $\hat{\Phi}$ *is* positive *if* $\mathsf{inv}(\hat{\Phi})$ *is even, and* negative *otherwise. We denote positive and negative contexts respectively by* $\hat{\Phi}^+$ *and* $\hat{\Phi}^-$*.*

In order to formulate the equivalent of the Iteration rule of EGs for flowers, we introduce a *pollination* relation that captures the availability of a flower in a given context:

▶ **Definition 6.3** (Pollination). *We say that a flower* ϕ *can be* pollinated *in a context* $\hat{\Phi}$*, written* $\phi \succ \hat{\Phi}$*, when there exists a bouquet* Ψ *with* $\phi \in \Psi$ *and contexts* $\hat{\Xi}$ *and* $\hat{\Xi}_0$ *s.t. either:*
Cross-pollination $\hat{\Phi} = \hat{\Xi}\{\Psi, \hat{\Xi}_0\}$*;*
Self-pollination $\hat{\Phi} = \hat{\Xi}\{\mathbf{x} \cdot \Psi \rhd \mathbf{y} \cdot \hat{\Xi}_0; \Delta\}$ *for some* $\mathbf{x}, \mathbf{y}, \Delta$*.*
A bouquet Φ *can be* pollinated *in* $\hat{\Phi}$*, written* $\Phi \succ \hat{\Phi}$*, if* $\phi \succ \hat{\Phi}$ *for all* $\phi \in \Phi$*.*

Figure 6 illustrates the meaning of pollination as a relation of *justification* between locations: the blue dot marks the location of the justifying/pollinating occurrence of ϕ, and the red dots all the areas that it justifies/pollinates, and thus where ϕ is available for use. We distinguish two cases of cross-pollination and self-pollination, as botanists do when describing the reproduction of flowers. This distinction does not exist in classical EGs, because pistils and petals are both identified as instances of cuts[7].

▶ Remark 6.4. Incidentally, the pollination relation also explains the *scope* of variables. Indeed, one can interpet red dots in Figure 6 as the allowed *usage* points for the variable *bound* at the linked blue dot. This hints at a possible *type-theoretic* variant of the flower calculus where variables are also used for higher-order individuals, including flowers.

Proofs. The inference rules of the flower calculus are presented in Figure 7. Read from top to bottom, they correspond to traditional inference rules deducing a necessary conclusion from a valid premiss. But we will prefer their backward, *bottom-up* reading: then they can be seen as *rewriting* rules that reduce a goal to a sufficient premiss, just like in our illustration of the illative transformations of EGs in Figure 1. Also, all rules manipulate *bouquets*: this is seen more clearly in the *graphical* presentation of the rules in appendix (Figures 8 and 9).

[7] The same phenomenon is at work in SFL: cross-pollination and self-pollination can be seen as generalizing the *forward* and *backward* interaction connectives $\bigcirc$ and $\triangleright$ of intuitionistic SFL [13, 22], while the original formulation of SFL for classical linear logic had only one interaction connective $*$ [12]. Through the Curry-Howard-Lambek correspondence, this is also reminiscent of the adjunction between products ($\bigcirc$) and exponentials ($\triangleright$) in *cartesian closed categories*, as opposed to the natural isomorphism $(-)^*$ of $*$*-autonomous categories*.

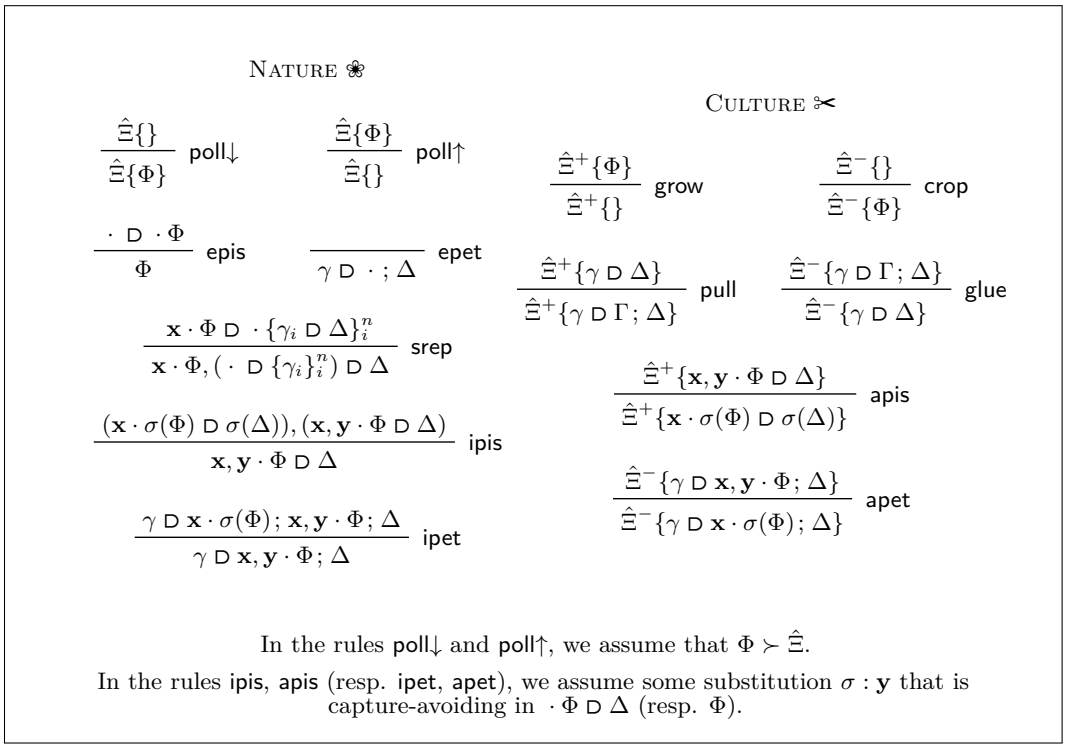


Figure 7 Rules of the flower calculus.

We partition the rules into two sets: the *natural* rules denoted by ❀ that apply in arbitrary contexts, and the *cultural* rules denoted by ✂ that apply exclusively in positive or negative contexts. In particular, every ❀-rule is both *analytic* (i.e. every atom in the premiss already appears in the conclusion) and *invertible* (see [21, Lemma B.17]); on the contrary, all ✂-rules are *non-invertible*, and they will be shown to be *admissible* in Section 8.

▶ **Definition 6.5** (Derivation). *Given a set of rules* R*, we write* $\Phi \to_{\mathsf{R}} \Psi$ *to indicate a rewrite* step *in* R*, that is an instance of some* $\mathsf{r} \in \mathsf{R}$ *with* Ψ *as premiss and* Φ *as conclusion. We just write* $\Phi \to \Psi$ *to mean* $\Phi \to_{❀ \cup ✂} \Psi$*. A* derivation $\Phi \to^n_{\mathsf{R}} \Psi$ *is a sequence of rewrite steps* $\Phi_0 \to_{\mathsf{R}} \Phi_1 \dots \to_{\mathsf{R}} \Phi_n$ *with* $\Phi_0 = \Phi$*,* $\Phi_n = \Psi$ *and* $n \geq 0$*. Generally the length* n *of the derivation does not matter, and we just write* $\Phi \to^*_{\mathsf{R}} \Psi$*. Finally, natural derivations are closed under arbitrary contexts: for every context* $\hat{\Xi}$*,* $\Phi \to_{❀} \Psi$ *implies* $\hat{\Xi}\{\Phi\} \to_{❀} \hat{\Xi}\{\Psi\}$*. We write* $\Phi \rightharpoonup_{❀} \Psi$ *to denote a* shallow *natural step, i.e. an instance of a* ❀*-rule in the empty context* $\Box$*.*

▶ **Definition 6.6** (Proof). *A* proof *of a bouquet* Φ *is a derivation* $\Phi \to^* \varnothing$*.*

In Peircean terms, the empty bouquet is the blank SA. Then proving a bouquet amounts to erasing it completely from SA, thus reducing it to trivial truth as in Figure 1. Figure 10 in appendix shows an example of ❀-proof in the flower calculus, both in textual and graphical syntax. Note that we used a non-duplicating version of the rules ipis and ipet, in order to save some space in the graphical presentation.

If we want to reason about *relative* truth, i.e. Φ is true under the assumption that Ψ is, we can simply rely on the existence of a derivation $\Phi \to^* \Psi$ in the full flower calculus. This will be justified by the soundness of all rules ([21, Theorem B.20]) as well as a *strong* completeness result (Corollary 8.8), that relies on the following strong deduction theorem:

▶ **Theorem 6.7** (Strong deduction). *$\Phi \to^* \Psi$ if and only if $\Psi \between \Phi \to^* \varnothing$.*

Contrary to full derivability, natural derivability $\Phi \to^*_{\text{❀}} \Psi$ is too weak to satisfy a strong deduction theorem. This is a consequence of the fact that ❀-rules are *invertible*, and thus can only relate equivalent bouquets. Indeed, as soon as $\Psi \between \Phi$ is ❀-provable but the converse $\Phi \between \Psi$ is not, it follows from the completeness of ❀-rules that Φ and Ψ are not equivalent: thus $\Phi \not\to^*_{\text{❀}} \Psi$, contradicting the strong deduction statement.

A trivial way to circumvent this is to define directly the relation of *hypothetical provability* $\Psi \vdash \Phi$ as $\Psi \between \Phi \to^* \varnothing$. This is closer to what one would find in sequent calculus, where hypothetical proofs are closed derivations of hypothetical sequents, not open derivations. The difference is that sequents capture only the *first-order*[8] implicative structure of logic, while flowers capture the full structure of intuitionistic FOL. This allows for a nice generalization of the notion of hypothetical provability, which will be useful in our completeness proof:

▶ **Definition 6.8.** *We say that Φ is* hypothetically provable *from Ψ in a fragment R of rules, written $\Psi \vdash_{\mathsf{R}} \Phi$, if $\hat{\Xi}\{\Phi\} \to^*_{\mathsf{R}} \hat{\Xi}\{\}$ for every context $\hat{\Xi}$ such that $\Psi \succ \hat{\Xi}$. We write $\Psi \vdash \Phi$ to denote hypothetical provability in the full flower calculus.*

▶ **Theorem 6.9** (Deduction). *$\Psi \vdash_{\text{❀}} \Phi$ if and only if $\vdash_{\text{❀}} \Psi \between \Phi$.*

7 Semantics

We now give a semantics to flowers in Kripke structures. We recall the standard definitions:

▶ **Definition 7.1.** *A* first-order structure *is a pair $(M, [\![\cdot]\!])$ where M is a non-empty set called the* domain*, and $[\![\cdot]\!]$ is a map called the* interpretation *that associates to each predicate symbol $p \in \mathcal{P}$ a relation $[\![p]\!] \subseteq M^{\mathsf{ar}(p)}$.*

▶ **Definition 7.2.** *A* Kripke structure *is a triplet $\mathcal{K} = (W, \leq, (M_w)_{w \in W})$, where W is the set of* worlds*, $\leq$ is a pre-order on W called* accessibility*, and $(M_w)_{w \in W}$ is a family of first-order structures indexed by W. Furthermore, we require the following monotonicity conditions to hold whenever $w \leq w'$: 1. $M_w \subseteq M_{w'}$; 2. for every $p \in \mathcal{P}$, $[\![p]\!]_w \subseteq [\![p]\!]_{w'}$.*

▶ **Definition 7.3.** *Given a Kripke structure $\mathcal{K}$ and a world w in $\mathcal{K}$, a* w-evaluation *is a function $e : \mathcal{V} \to M_w$. The interpretation map of M_w is extended to variables and substitutions with respect to any w-evaluation e as follows:*

$$[\![x]\!]_e = e(x) \qquad\qquad [\![\sigma]\!]_e(x) = [\![\sigma(x)]\!]_e$$

The crux of Kripke semantics is the *forcing* relation, that captures the truth-conditions of statements in Kripke structures. While it is usually defined on formulas, here we adapt the definition to flowers, which in our opinion makes it simpler and more uniform since flowers can be seen as built from essentially one big constructor:

▶ **Definition 7.4.** *The* depth *$|-|$ of a flower/garden is defined by mutual recursion:*

$$|p(\vec{\mathbf{x}})| = 0 \qquad |\mathbf{x} \cdot \Phi| = \max_{\phi \in \Phi} |\phi| \qquad |\gamma \between \Delta| = 1 + \max(|\gamma|, \max_{\delta \in \Delta} |\delta|)$$

▶ **Definition 7.5.** *Given some Kripke structure $\mathcal{K}$, the* forcing *relation $w \Vdash \phi\,[e]$ between a world w, a flower ϕ and a w-evaluation e is defined by induction on $|\phi|$ as follows:*

[8] As opposed to *higher-order*, in the sense of having negatively nested implications.

Atom $w \Vdash p(\vec{\mathbf{x}})\,[e]$ *iff* $[\![\vec{\mathbf{x}}]\!]_e \in [\![p]\!]_w$*;*

Flower $w \Vdash \mathbf{x} \cdot \Phi \between \{\mathbf{x}_i \cdot \Phi_i\}_i^n\,[e]$ *iff for every* $w' \geq w$ *and every* w'*-evaluation* e'*, if* $w' \Vdash \Phi\,[e|_{\mathbf{x}} e']$ *then there is some* $1 \leq i \leq n$ *and* w'*-evaluation* e'' *such that* $w' \Vdash \Phi_i\,[e|_{\mathbf{x}} e'|_{\mathbf{x}_i} e'']$*.*

Bouquet $w \Vdash \Phi\,[e]$ *iff* $w \Vdash \phi\,[e]$ *for every* $\phi \in \Phi$*.*

Lastly, we define the notion of *semantic entailment* $\Phi \vDash \Psi$ on bouquets, mirroring the syntactic entailment $\Phi \vdash \Psi$ of the last section:

▶ **Definition 7.6.** *Let* $\mathcal{K}$ *be a Kripke structure, and* Φ, Ψ *some bouquets. We say that* Φ semantically entails Ψ *in* $\mathcal{K}$*, written* $\Phi \vDash_{\mathcal{K}} \Psi$*, when* $w \Vdash \Phi\,[e]$ *implies* $w \Vdash \Psi\,[e]$ *for every world* $w \in W$ *and* w*-evaluation* e*. This entailment is* valid *if it holds for any Kripke structure* $\mathcal{K}$*, and in that case we simply write* $\Phi \vDash \Psi$*. We say that* Φ *is* semantically equivalent *to* Ψ*, written* $\Phi \dashv\vDash \Psi$*, when* $\Phi \vDash \Psi$ *and* $\Psi \vDash \Phi$*.*

8 Completeness

We now outline a direct completeness proof for the natural fragment ❀ of the flower calculus: every true flower ϕ is naturally provable, i.e. $\vDash \phi$ implies $\vdash_{❀} \phi$. Since this fragment is analytic, we cannot reuse most completeness proofs from the literature, because they usually rely on a non-analytic principle like the cut rule of sequent calculus. Our insight was to adapt techniques from the *semantic cut-elimination* proof given by Hermant in [28], which is nonetheless relatively close to the original completeness proof of Gödel. A novelty of our proof is that it dispenses completely with the need for *Henkin witnesses*.

First we need to generalize our notions of syntactic and semantic entailment to possibly *infinite* sets of flowers, so-called *theories*:

▶ **Definition 8.1.** *Any set* $\mathcal{T} \subseteq \mathbb{F}$ *of flowers is called a* theory*. In particular, a bouquet can be regarded as a finite theory, by forgetting the number of repetitions of its elements. We say that a bouquet* Φ *is* provable from *a theory* $\mathcal{T}$*, written* $\mathcal{T} \vdash \Phi$*, if there exists a bouquet* $\Psi \subseteq \mathcal{T}$ *such that* $\Psi \vdash \Phi$*. Given a Kripke structure* $\mathcal{K}$*, a world* w *in* $\mathcal{K}$ *and a* w*-evaluation* e*, we say that* $\mathcal{T}$ *is* forced *by* w *under* e*, written* $w \Vdash \mathcal{T}\,[e]$*, if* $w \Vdash \phi\,[e]$ *for all* $\phi \in \mathcal{T}$*. Then* Φ *is a* consequence *of* $\mathcal{T}$*, written* $\mathcal{T} \vDash_{\mathcal{K}} \Phi$*, if* $w \Vdash \mathcal{T}\,[e]$ *implies* $w \Vdash \Phi\,[e]$ *for every world* w *in* $\mathcal{K}$ *and* w*-evaluation* e*.*

▶ **Definition 8.2.** *A theory* $\mathcal{T}$ *is said to be* ψ-consistent *when* $\mathcal{T} \nvdash_{❀} \psi$*, and* ψ-complete *when for all* $\phi \in \mathbb{F}$*, either* $\mathcal{T}, \phi \vdash_{❀} \psi$ *or* $\phi \in \mathcal{T}$*.*

Intuitively, a theory $\mathcal{T}$ is ψ-consistent when one cannot deduce ψ from it, and ψ-complete when it *decides* any formula ϕ relatively to ψ. This is better understood by considering the special case where $\psi = (\between)$ is the *absurd* flower: then consistency means that one cannot derive any contradiction from $\mathcal{T}$; and completeness that $\mathcal{T}$ either *refutes* ϕ syntactically with a proof of $\Phi, \phi \between (\between)$ for some $\Phi \subseteq \mathcal{T}$, or already *validates* it "semantically", i.e. without the need for a proof since $\phi \in \mathcal{T}$.

The next two propositions constitute the central argument that allows the completeness proof to go through despite the analyticity of ❀-rules. They are a direct adaptation of [28, Proposition 7], which Hermant identifies as "an important property of any A-consistent, A-complete theory, [...] that it enjoys some form of the subformula property".

Roughly, the first proposition captures the intuitionistic truth-conditions that make a flower *valid* (i.e. true in every model) by modelling them on material implication, just like Peirce would do with his scroll (see Section 3): ϕ is true if the content Φ_i of one of its petals (consequents) is, or if the content Φ of its pistil (antecedent) is not.

▶ **Proposition 8.3** (Analytic truth). *Let $\psi \in \mathbb{F}$, $\mathcal{T}$ some ψ-consistent and ψ-complete theory, and $\phi = \mathbf{x} \cdot \Phi \rhd \Delta$ with $\Delta = \{\delta_i\}_i^n = \{\mathbf{x}_i \cdot \Phi_i\}_i^n$ such that $\phi \in \mathcal{T}$. Then for every substitution $\sigma : \mathbf{x}$, either $\sigma(\Phi_i) \subseteq \mathcal{T}$ for some $1 \leq i \leq n$, or $\mathcal{T} \nvdash_{❀} \sigma(\Phi)$.*

Proof. Suppose the contrary, i.e. there is a substitution σ such that $\mathcal{T} \vdash_{❀} \sigma(\Phi)$ and for all $1 \leq i \leq n$, there is some $\phi_i \in \Phi_i$ ① such that $\sigma(\phi_i) \notin \mathcal{T}$. Thus by ψ-completeness of $\mathcal{T}$, we get $\mathcal{T}, \sigma(\phi_i) \vdash_{❀} \psi$. So there are $\Psi \subseteq \mathcal{T}$ and $\Psi_i \subseteq \mathcal{T} \cup \sigma(\phi_i)$ such that $\Psi \vdash_{❀} \sigma(\Phi)$ ② and $\Psi_i \vdash_{❀} \psi$ ③. Now it cannot be the case that $\Psi_i \subseteq \mathcal{T}$, otherwise by weakening and ψ-consistency of $\mathcal{T}$ we would have $\Psi_i \nvdash_{❀} \psi$. So there must exist $\Psi_i' \subseteq \mathcal{T}$ such that $\Psi_i = \Psi_i' \cup \sigma(\phi_i)$ ④. Again by weakening and ψ-consistency of $\mathcal{T}$, we get $\Psi, \bigcup_{i=1}^n \Psi_i', \phi \nvdash_{❀} \psi$. Now we derive a contradiction by showing $\Psi, \bigcup_{i=1}^n \Psi_i', \phi \vdash_{❀} \psi$. Let $\hat{\Xi}$ be a context such that $\Psi, \bigcup_{i=1}^n \Psi_i', \phi \succ \hat{\Xi}$ ⑤. Then $\hat{\Xi}\{\psi\} \to_{❀}^* \hat{\Xi}\{\}$ with the following derivation:

$$
\begin{array}{lll}
\hat{\Xi}\{\psi\} & \to_{\mathsf{epis}} & \hat{\Xi}\{\cdot \rhd \cdot \psi\} \\
& \to_{\mathsf{poll}\uparrow} & \hat{\Xi}\{\cdot \phi \rhd \cdot \psi\} \qquad (⑤) \\
& \to_{\mathsf{ipis}} & \hat{\Xi}\{\cdot (\cdot \sigma(\Phi) \rhd \sigma(\Delta)), \phi \rhd \cdot \psi\} \\
& \to_{\mathsf{poll}\downarrow} & \hat{\Xi}\{\cdot (\cdot \rhd \sigma(\Delta)), \phi \rhd \cdot \psi\} \qquad (②, ⑤) \\
& \to_{\mathsf{srep}} & \hat{\Xi}\{\cdot \phi \rhd \cdot \{\sigma(\delta_i) \rhd \cdot \psi\}_i^n\} \\
& = & \hat{\Xi}\{\cdot \phi \rhd \cdot \{\mathbf{x}_i \cdot \sigma(\Phi_i) \rhd \cdot \psi\}_i^n\} \\
& \to_{\mathsf{poll}\downarrow}^n & \hat{\Xi}\{\cdot \phi \rhd \cdot \{\mathbf{x}_i \cdot \sigma(\Phi_i) \rhd \cdot\}_i^n\} \qquad (①, ③, ④, ⑤) \\
& \to_{\mathsf{epet}}^n & \hat{\Xi}\{\cdot \phi \rhd \cdot\} \\
& \to_{\mathsf{epet}} & \hat{\Xi}\{\}
\end{array}
$$

◀

Dually, the second proposition captures the grounds on which a flower can be deemed *invalid* (i.e. false in at least one model): ϕ is not true if assuming that its pistil Φ is true is not sufficient to conclude that one of its petals Φ_i is.

▶ **Proposition 8.4** (Analytic refutation). *Let $\psi \in \mathbb{F}$, $\mathcal{T}$ some ψ-consistent and ψ-complete theory, and $\phi = \mathbf{x} \cdot \Phi \rhd \Delta$ with $\Delta = \{\delta_i\}_i^n = \{\mathbf{x}_i \cdot \Phi_i\}_i^n$ such that $\mathcal{T} \nvdash_{❀} \phi$. Then for every $1 \leq i \leq n$ and substitution $\sigma : \mathbf{x}_i$, there is some $\phi_i \in \Phi_i$ such that $\mathcal{T}, \Phi \nvdash_{❀} \sigma(\phi_i)$.*

Proof. Suppose the contrary, i.e. there are some $1 \leq i \leq n$ and $\sigma : \mathbf{x}_i$ such that $\mathcal{T}, \Phi \vdash_{❀} \sigma(\Phi_i)$. Therefore there must exist $\Psi \subseteq \mathcal{T}$ and $\Phi_0 \subseteq \Phi$ ① such that $\Psi, \Phi_0 \vdash_{❀} \sigma(\Phi_i)$ ②. By hypothesis, for every $\Phi' \subseteq \mathcal{T}$ there is a context $\hat{\Xi}$ such that $\Phi' \succ \hat{\Xi}$ and $\hat{\Xi}\{\phi\} \nrightarrow_{❀}^* \hat{\Xi}\{\}$. We now derive a contradiction by showing $\hat{\Xi}\{\phi\} \to_{❀}^* \hat{\Xi}\{\}$ for all $\hat{\Xi}$ such that $\Psi \succ \hat{\Xi}$ ③:

$$
\begin{array}{lll}
\hat{\Xi}\{\phi\} & \to_{\mathsf{ipet}} & \hat{\Xi}\{\mathbf{x} \cdot \Phi \rhd \cdot \sigma(\Phi_i); \Delta\} \\
& \to_{\mathsf{poll}\downarrow} & \hat{\Xi}\{\mathbf{x} \cdot \Phi \rhd \cdot ; \Delta\} \qquad (①, ②, ③) \\
& \to_{\mathsf{epet}} & \hat{\Xi}\{\}
\end{array}
$$

◀

Next, we define the so-called *universal Kripke structure* ✿(ψ) relative to a flower ψ:

▶ **Definition 8.5.** *Let $\psi \in \mathbb{F}$. The* universal Kripke structure ✿(ψ) *has:*

- *The set of ψ-consistent and ψ-complete theories as its worlds;*
- *Set inclusion $\subseteq$ as its accessibility relation;*

- *For each world $\mathcal{T}$, a first-order structure whose domain is the set of variables $\mathcal{V}$, and whose interpretation map is given by $[\![p]\!]_{\mathcal{T}} = \{\vec{\mathbf{x}} \mid p(\vec{\mathbf{x}}) \in \mathcal{T}\}$.*

One can easily check that the monotonicity conditions of Kripke structures hold for $✿(\psi)$.

We are now equipped to formulate the main *adequacy* lemma, which relates forcing in $✿(\psi)$ to ψ-consistency and ψ-completeness:

▶ **Lemma 8.6** (Adequacy). *Let $\phi, \psi \in \mathbb{F}$, $\mathcal{T}$ a ψ-consistent and ψ-complete theory, and σ a substitution. Then 1. $\sigma(\phi) \in \mathcal{T}$ implies $\mathcal{T} \Vdash \phi\,[\sigma]$, and 2. $\mathcal{T} \nvdash_{❀} \sigma(\phi)$ implies $\mathcal{T} \nVdash \phi\,[\sigma]$.*

Proof. The proof goes by induction on $|\phi|$. We only give an informal sketch, see [21, Appendix C.2] for the detailed proof. There are just two cases to consider:

Base case $\phi = p(\vec{\mathbf{x}})$. The first statement is trivial. The second statement is immediate from reflexivity and weakening lemmas on the hypothetical provability relation $\vdash$.

Recursive case $\phi = \gamma \multimap \Delta$. The first statement follows from Proposition 8.3. The second statement follows from Proposition 8.4, as well as the existence and properties of the completion procedure. ◀

We get the completeness theorem as a near-direct consequence:

▶ **Theorem 8.7** (Completeness). $\Phi \vDash \Psi$ *implies* $\Phi \vdash_{❀} \Psi$.

Combined with strong deduction (Theorem 6.7), this also yields a strong completeness theorem for the full flower calculus[9]:

▶ **Corollary 8.8** (Strong completeness). $\Phi \vDash \Psi$ *implies* $\Psi \to^* \Phi$.

Finally, the composition of the soundness, completeness and deduction theorems ([21, Theorem B.20], Theorem 8.7 and Theorem 6.9) gives the admissibility of ✂-rules, and thus the analyticity of the flower calculus:

▶ **Corollary 8.9** (Cult-elimination). *If $\Phi \vdash \Psi$ then $\Phi \vdash_{❀} \Psi$.*

9 Related works

Intuitionistic EGs. We have already mentioned the seminal work of Oostra, who introduced in [46] an intuitionistic version of Alpha. In [47] he describes its natural extension with LoIs to get an intuitionistic version of Beta, and in [48] he gives formal soundness and completeness proofs for intuitionistic Alpha, based on a linear notation for graphs. Ma and Pietarinen have developed in [40] their own system of intuitionistic EGs for propositional logic, with a different set of inference rules than Oostra's. They give a more systematic proof theory, including deduction, soundness and completeness theorems with respect to Heyting algebras.

Our work brings several new contributions on top of those:

Variadicity Our multiset-based definition of flowers captures faithfully the *variadic* nature of juxtaposition and n-ary scrolls in the diagrammatic syntax. In contrast, previous formalizations rely on a restricted inductive syntax which only captures graphs that are isomorphic to formulas built with binary connectives.

Intuitionistic binders While replacing LoIs with binders and variables has already been done by Sowa in the context of classical EGs [56], it seems like we are the first to adapt the idea to the intuitionistic setting.

[9] Actually it already works for the fragment $❀ \cup \{\mathsf{grow}\}$, thanks to the proof of the strong deduction theorem (see [21, Appendix C.1]).

Analyticity To our knowledge, we are the first to give a Kripke semantics to a syntax based on EGs, and to use this to obtain an analyticity result[10].

Invertibility The natural fragment of the flower calculus appears to be the first proof system based on EGs where all rules are *invertible*.

Deep inference. While the deep inference literature is most furnished with systems for classical logic, a few works tackle intuitionistic logics: the seminal work of Tiu, who proposed a *calculus of structures* for intuitionistic FOL [59], was followed by computational interpretations of the implicative fragment in Guenot's thesis [26]. There are also nested sequent systems for (propositional) full intuitionistic linear logic [15], standard and constant-domain intuitionistic FOL [23], and intuitionistic modal logics [14, 32, 37]. The flower calculus is closer to Guenot's nested sequent calculi for implicative logic which also function as rewriting systems, but extends them to full intuitionistic FOL.

Labelled sequent calculi. For a long time, it was believed that there could not be fully invertible proof systems for intuitionistic logics, even in the propositional case. While this might be true in standard Gentzen formalisms, recent works have shown that it is possible in the context of *labelled sequent calculi*: first with Lyon's G3IntQ calculus for FOL [36, Section 3.3], and then with the calculus labIS4$_{\leq}$ of Girlando et al. for the modal logic S4 [25]. In these systems, invertibility is made possible by the addition of *semantic* information to sequents, in the form of so-called *labels* and *relational atoms* that respectively encode the worlds and accessibility relations of Kripke structures. The flower calculus follows instead a purely *syntactic* approach, by relying on deep inference to retrieve what would normally be semantic information from the context $\hat{\Xi}$ in the pollination rules poll$\uparrow$ and poll$\downarrow$.

Categorical EGs. Since the seminal work of Brady and Trimble in 2000 on the formalization of EGs in category theory [7, 8], there have been various efforts to find rich categorical axiomatizations of Beta. The first approach – initiated in [8] – is based on *string diagrams*, and has recently enabled strong connections with *Frobenius algebras* and *bicategories or relations* [43, 27, 6]. A second approach makes use of the concept of *generic figure* [11], introduced by Reyes as a basic building block for *topos theory* [33]. We do not know however of any attempt to uncover the categorical structures underlying intuitionistic EGs. The flower calculus might be an interesting candidate, in that the invertibility of the natural fragment could enable a purely *equational* approach.

Coherent logic. We noticed a formal connection between flowers and *coherent logic*, a subset of the formulas of FOL discovered by Skolem in 1920 [55] that is capable of expressing many mathematical theories, and has close connections to topos theory [31, Section D3.3]. Indeed, the interpretation $\lfloor \mathbf{x} \cdot \Phi \rhd \Delta \rfloor$ of a generic flower is given by the following formula, which has exactly the shape of a coherent formula as described e.g. in [5]:

$$\forall \mathbf{x}. \left(\bigwedge_{\phi \in \Phi} \lfloor \phi \rfloor \supset \bigvee_{\mathbf{y} \cdot \Psi \in \Delta} \exists \mathbf{y}. \bigwedge_{\psi \in \Psi} \lfloor \psi \rfloor \right)$$

[10] Ma and Pietarinen claim in [39] that Alpha is analytic because it can simulate the cut rule of sequent calculus. This is a misinterpretation, since this supports precisely the *contrary*: the ability to simulate the cut rule with a constant number of rules implies the *non-analyticity* of one the rules involved (namely, Peirce's Deletion rule). Still, the notion of analyticity is not yet fully understood in deep inference systems, as discussed in [10].

The only difference is that flowers can be *nested*, while coherent formulas (also called coherent *sequents*) are first-order, in the sense that ϕ and ψ must be atoms. Coherent formulas appear in the theory of *focusing* in sequent calculi [41], and they lend themselves to simple proof search procedures that allow for *explainable proof automation* in ITPs [5, 30]. A higher-order variant of coherent formulas that is almost isomorphic to flowers has also been used to construct an intuitionistic version of the *arithmetical hierarchy*, as well as a fully *non-invertible* proof system for propositional intuitionistic logic [9].

Graph calculi. In the last twenty years, Veloso et al. have studied a series of so-called *graph calculi*, where first-order relations are represented by graphs in the sense of graph theory, and inference rules as graph transformations. The first graph calculus was introduced informally by Curtis and Lowe in 1996 [16], as a graphical notation supposedly capturing both relational calculus, and the sequential calculus of Karger and Hoare [63]. Veloso et al. gave sound and complete syntax and semantics to the calculus in [17], showing that it captures *positive* first-order logic on *binary* relations. They then extended their formalism to support relational complementation (negation) [18] as well as various modal [62, 60] and dynamic logics [61].

Graph calculi only handle *binary* relations and classical logic, while EGs and the flower calculus support relations of arbitrary arity and intuitionistic logic. We conjecture that the relationship between graph calculi and EGs is similar to that between *commutative diagrams* and *string diagrams* in category theory: the former represent relations/morphisms as edges between individuals/objects, while the latter represent them dually as points related by lines. EGs could then be understood as a *hypergraph* generalization of graph calculi, where lines of identity are hyperedges connecting multiple predicate vertices.

Development calculi. Through their backward reading, the rules of the flower calculus can be understood as primitive *tactics* for building proofs interactively. In [2, Chapter 3], Ayers calls such systems *development calculi*. In particular, he presents his own development calculus inspired by McBride's OLEG system [42] and Ganesalingam & Gowers's prover [24] called the `Box` calculus, where goals are represented by a so-called `Box` data structure very similar to flowers. In particular, `Box`es have so-called *disjunctive pairs* to reduce backtracking, that correspond to the petals of flowers. The main difference is that the `Box` calculus is based on dependent type theory instead of FOL: this allows to store the partial proof terms inside of the `Box`es themselves, while this information is lost during the construction of flowers. However, there is no completeness nor analyticity result for the `Box` calculus. It would be interesting to investigate further connections, in order to develop a dependently-typed version of the flower calculus.

References

1 Samson Abramsky and Achim Jung. *Domain Theory*. Oxford University Press, Inc., USA, 1995.

2 Edward W. Ayers. *A Tool for Producing Verified, Explainable Proofs*. PhD thesis, University of Cambridge, 2021.

3 Edward W. Ayers, Mateja Jamnik, and W. T. Gowers. A Graphical User Interface Framework for Formal Verification. In Liron Cohen and Cezary Kaliszyk, editors, *12th International Conference on Interactive Theorem Proving (ITP 2021)*, volume 193 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:16, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.ITP.2021.4`.

4 Yves Bertot, Gilles Kahn, and Laurent Théry. Proof by pointing. In Masami Hagiya and John C. Mitchell, editors, *Theoretical Aspects of Computer Software*, volume 789, pages 141–160. Springer Berlin Heidelberg, 1994. Series Title: Lecture Notes in Computer Science. doi:10.1007/3-540-57887-0_94.

5 Marc Bezem and Thierry Coquand. Automating Coherent Logic. In Geoff Sutcliffe and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, Lecture Notes in Computer Science, pages 246–260, Berlin, Heidelberg, 2005. Springer. doi:10.1007/11591191_18.

6 Filippo Bonchi, Alessandro Di Giorgio, Nathan Haydon, and Pawel Sobocinski. Diagrammatic Algebra of First Order Logic, January 2024. arXiv:2401.07055 [cs, math]. doi:10.48550/arXiv.2401.07055.

7 Geraldine Brady and Todd H. Trimble. A categorical interpretation of C.S. Peirce's propositional logic Alpha. *Journal of Pure and Applied Algebra*, 149(3):213–239, June 2000. doi:10.1016/S0022-4049(98)00179-0.

8 Geraldine Brady and Todd H. Trimble. A String Diagram Calculus for Predicate Logic and C. S. Peirce's System Beta, June 2000. URL: https://ncatlab.org/nlab/files/BradyTrimbleString.pdf.

9 Taus Brock-Nannestad and Danko Ilik. An Intuitionistic Formula Hierarchy Based on High-School Identities. *Mathematical Logic Quarterly*, 65(1):57–79, May 2019. arXiv: 1601.04876. doi:10.1002/malq.201700047.

10 Paola Bruscoli and Alessio Guglielmi. On Analyticity in Deep Inference. *Mathematical Structures in Computer Science*, 29(Special Issue 8: A special issue on structural proof theory, automated reasoning and computation in celebration of Dale Miller's 60th birthday), 2019. doi:10.1017/S0960129519000136.

11 Gianluca Caterina and Rocco Gangle. A New Syntax for Diagrammatic Logic: A Generic Figures Approach. In Yaroslav D. Sergeyev and Dmitri E. Kvasov, editors, *Numerical Computations: Theory and Algorithms*, Lecture Notes in Computer Science, pages 43–58, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-40616-5_4.

12 Kaustuv Chaudhuri. Subformula linking as an interaction method. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Interactive Theorem Proving*, volume 7998, pages 386–401. Springer Berlin Heidelberg, 2013. Series Title: Lecture Notes in Computer Science. doi:10.1007/978-3-642-39634-2_28.

13 Kaustuv Chaudhuri. Subformula linking for intuitionistic logic with application to type theory. In André Platzer and Geoff Sutcliffe, editors, *Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings*, volume 12699 of *Lecture Notes in Computer Science*, pages 200–216. Springer, 2021. doi:10.1007/978-3-030-79876-5_12.

14 Kaustuv Chaudhuri, Sonia Marin, and Lutz Straßburger. Modular focused proof systems for intuitionistic modal logics. In *International Conference on Formal Structures for Computation and Deduction*, 2016.

15 Ranald Clouston, Jeremy Dawson, Rajeev Goré, and Alwen Tiu. Annotation-Free Sequent Calculi for Full Intuitionistic Linear Logic. In Simona Ronchi Della Rocca, editor, *Computer Science Logic 2013 (CSL 2013)*, volume 23 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 197–214, Dagstuhl, Germany, 2013. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CSL.2013.197.

16 Sharon Curtis and Gavin Lowe. Proofs with graphs. *Science of Computer Programming*, 26(1):197–216, May 1996. doi:10.1016/0167-6423(95)00025-9.

17 Renata De Freitas, Paulo A. S. Veloso, Sheila R. M. Veloso, and Petrucio Viana. On graph reasoning. *Information and Computation*, 207(10):1000–1014, October 2009. doi:10.1016/j.ic.2008.11.004.

18 Renata De Freitas, Paulo A. S. Veloso, Sheila R. M. Veloso, and Petrucio Viana. A Calculus for Graphs with Complement. In Ashok K. Goel, Mateja Jamnik, and N. Hari Narayanan, editors, *Diagrammatic Representation and Inference*, volume 6170, pages 84–98. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. Series Title: Lecture Notes in Computer Science. doi:10.1007/978-3-642-14600-8_11.

19 Pablo Donato. flowers-metatheory, November 2022. Software, swhId: swh:1:dir:290076a847ca95e93c17fb66659086d7f68be014 (visited on 2024-06-20). URL: https://github.com/Champitoad/flowers-metatheory.

20 Pablo Donato. flower-prover, October 2023. Software, swhId: swh:1:dir:fcc934cae3a75692c031dc82ffdab138084a472d (visited on 2024-06-20). URL: https://github.com/Champitoad/flower-prover.

21 Pablo Donato. The Flower Calculus. Preprint, April 2024. URL: https://arxiv.org/abs/2402.15174.

22 Pablo Donato, Pierre-Yves Strub, and Benjamin Werner. A drag-and-drop proof tactic. In *Proceedings of the 11th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2022, pages 197–209, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3497775.3503692.

23 Melvin Fitting. Nested Sequents for Intuitionistic Logics. *Notre Dame Journal of Formal Logic*, 55(1):41–61, 2014. doi:10.1215/00294527-2377869.

24 M. Ganesalingam and W. T. Gowers. A Fully Automatic Theorem Prover with Human-Style Output. *Journal of Automated Reasoning*, 58(2):253–291, February 2017. doi:10.1007/s10817-016-9377-1.

25 Marianna Girlando, Roman Kuznets, Sonia Marin, Marianela Morales, and Lutz Straßburger. Intuitionistic S4 is decidable. In *2023 38th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13, June 2023. arXiv:2304.12094 [cs]. doi:10.1109/LICS56636.2023.10175684.

26 Nicolas Guenot. *Nested Deduction in Logical Foundations for Computation*. PhD thesis, Ecole Polytechnique X, April 2013. URL: https://pastel.archives-ouvertes.fr/pastel-00929908.

27 Nathan Haydon and Paweł Sobociński. Compositional Diagrammatic First-Order Logic. In Ahti-Veikko Pietarinen, Peter Chapman, Leonie Bosveld-de Smet, Valeria Giardino, James Corter, and Sven Linker, editors, *Diagrammatic Representation and Inference*, volume 12169, pages 402–418. Springer International Publishing, Cham, 2020. Series Title: Lecture Notes in Computer Science. doi:10.1007/978-3-030-54249-8_32.

28 Olivier Hermant. Semantic Cut Elimination in the Intuitionistic Sequent Calculus. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Dough Tygar, Moshe Y. Vardi, Gerhard Weikum, and Paweł Urzyczyn, editors, *Typed Lambda Calculi and Applications*, volume 3461, pages 221–233. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. Series Title: Lecture Notes in Computer Science. doi:10.1007/11417170_17.

29 John Howse, Gem Stapleton, and John Taylor. Spider Diagrams. *LMS Journal of Computation and Mathematics*, 8:145–194, January 2005. Publisher: Cambridge University Press. doi:10.1112/S1461157000000942.

30 Predrag Janičić and Julien Narboux. Automated generation of illustrated proofs in geometry and beyond. *Annals of Mathematics and Artificial Intelligence*, July 2023. doi:10.1007/s10472-023-09857-y.

31 Peter T. Johnstone. *Sketches of an Elephant: A Topos Theory Compendium*, volume 2 of *Oxford Logic Guides*. Clarendon Press, Oxford, England, September 2002.

32 Roman Kuznets and Lutz Straßburger. Maehara-style modal nested calculi. *Archive for Mathematical Logic*, 58(3-4):359–385, May 2019. doi:10.1007/s00153-018-0636-1.

33 Marie La Palme Reyes, Gonzalo E. Reyes, and Houman Zolfaghari. *Generic figures and their glueings*. Polimetrica, International Scientific Publisher, 2008.

34 C. I. Lewis. A survey of symbolic logic. *Journal of Philosophy, Psychology and Scientific Methods*, 17(3):78–79, 1920. doi:10.2307/2940631.

35 Sven Linker, Jim Burton, and Mateja Jamnik. Tactical Diagrammatic Reasoning. *Electronic Proceedings in Theoretical Computer Science*, 239:29–42, January 2017. doi:10.4204/EPTCS.239.3.

36 Tim Lyon. *Refining Labelled Systems for Modal and Constructive Logics with Applications*. PhD thesis, Vienna University of Technology, July 2021. doi:10.48550/arXiv.2107.14487.

37 Tim S. Lyon. Nested Sequents for Intuitionistic Modal Logics via Structural Refinement. In Anupam Das and Sara Negri, editors, *Automated Reasoning with Analytic Tableaux and Related Methods*, Lecture Notes in Computer Science, pages 409–427, Cham, 2021. Springer International Publishing. doi:10.1007/978-3-030-86059-2_24.

38 Thomas Långbacka, Rimvydas Rukšėnas, and Joakim von Wright. TkWinHOL: A tool for Window Inference in HOL. In E. Thomas Schubert, Philip J. Windley, and James Alves-Foss, editors, *Higher Order Logic Theorem Proving and Its Applications*, Lecture Notes in Computer Science, pages 245–260, Berlin, Heidelberg, 1995. Springer. doi:10.1007/3-540-60275-5_69.

39 Minghui Ma and Ahti-Veikko Pietarinen. Proof Analysis of Peirce's Alpha System of Graphs. *Studia Logica*, 105(3):625–647, June 2017. doi:10.1007/s11225-016-9703-y.

40 Minghui Ma and Ahti-Veikko Pietarinen. A graphical deep inference system for intuitionistic logic. *Logique et Analyse*, 245:73–114, January 2019. doi:10.2143/LEA.245.0.3285706.

41 Sonia Marin, Dale Miller, Elaine Pimentel, and Marco Volpe. From axioms to synthetic inference rules via focusing. *Annals of Pure and Applied Logic*, 173(5):103091, May 2022. doi:10.1016/j.apal.2022.103091.

42 Conor McBride. *Dependently Typed Functional Programs and their Proofs*. PhD thesis, University of Edinburgh, July 2000. URL: https://era.ed.ac.uk/handle/1842/374.

43 Paul-André Melliès and Noam Zeilberger. A bifibrational reconstruction of Lawvere's presheaf hyperdoctrine. *arXiv:1601.06098 [cs, math]*, August 2016. arXiv:1601.06098.

44 Robin Milner. The use of machines to assist in rigorous proof. *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 312(1522):411–422, 1984. doi:10.1098/rsta.1984.0067.

45 Leonardo de Moura and Sebastian Ullrich. The Lean 4 Theorem Prover and Programming Language. In André Platzer and Geoff Sutcliffe, editors, *Automated Deduction – CADE 28*, pages 625–635, Cham, 2021. Springer International Publishing.

46 Arnold Oostra. *Los gráficos Alfa de Peirce aplicados a la lógica intuicionista*. Number 2 in Cuadernos de Sistemática Peirceana. Centro de Sistemática Peirceana, 2010.

47 Arnold Oostra. *Gráficos existenciales Beta intuicionistas*. Number 3 in Cuadernos de Sistemática Peirceana. Centro de Sistemática Peirceana, 2011.

48 Arnold Oostra. Equivalence proof for intuitionistic existential alpha graphs. In *Diagrammatic Representation and Inference: 12th International Conference, Diagrams 2021, Virtual, September 28–30, 2021, Proceedings*, pages 188–195, Berlin, Heidelberg, 2021. Springer-Verlag. doi:10.1007/978-3-030-86062-2_16.

49 Arnold Oostra. *Advances in Peircean Mathematics: The Colombian School*, chapter Intuitionistic and Geometrical Extensions of Peirce's Existential Graphs, pages 105–180. De Gruyter, 2022.

50 Charles Sanders Peirce. Prolegomena to an Apology for Pragmaticism. *The Monist*, 16(4):492–546, 1906. Publisher: Oxford University Press. URL: https://www.jstor.org/stable/27899680.

51 Brian Ritchie. *The Design and Implementation of an Interactive Proof Editor*. PhD thesis, The University of Edinburgh, 1988. Accepted: 2013-04-02T15:18:04Z Publisher: The University of Edinburgh. URL: https://era.ed.ac.uk/handle/1842/6607.

52 Don D. Roberts. *The Existential Graphs of Charles S. Peirce*. De Gruyter Mouton, Berlin, Boston, 1973. doi:doi:10.1515/9783110226225.

53 Benoit Rognier and Guillaume Duhamel. Présentation de la plateforme edukera. In *Vingt-septièmes Journées Francophones des Langages Applicatifs (JFLA 2016)*, 2016.

54 Sun-Joo Shin. *The Iconic Logic of Peirce's Graphs*. The MIT Press, May 2002. doi:10.7551/mitpress/3633.001.0001.

55 Thoralf Skolem. Logisch-kombinatorische untersuchungen Über die erfüllbarkeit oder bewiesbarkeit mathematischer sätze nebst einem theorem Über dichte mengen. In Thoralf Skolem, editor, *Selected Works in Logic*. Universitetsforlaget, 1920.

56 John Sowa. Peirce's Tutorial on Existential Graphs. *Semiotica*, 186:345–394, 2011. doi:10.1515/semi.2011.060.

57 M. H. Stone. Topological representations of distributive lattices and Brouwerian logics. In *Časopis pro pěstování matematiky a fysiky*, volume 067, pages 1–25, 1938. ISSN: 1802-114X Issue: 1 Journal Abbreviation: Časopis Pěst. Mat. Fys. doi:10.21136/CPMF.1938.124080.

58 The Coq Development Team. The Coq Proof Assistant, 2022. doi:10.5281/zenodo.7313584.

59 Alwen Tiu. A Local System for Intuitionistic Logic. In Miki Hermann and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, Lecture Notes in Computer Science, pages 242–256, Berlin, Heidelberg, 2006. Springer. doi:10.1007/11916277_17.

60 Paulo Veloso, Sheila Veloso, and Mario Benevides. On Graph Calculi for Multi-modal Logics. *Electronic Notes in Theoretical Computer Science*, 312, April 2015. doi:10.1016/j.entcs.2015.04.014.

61 Paulo A. S. Veloso, Sheila R. M. Veloso, and Mario R. F. Benevides. PDL for structured data: a graph-calculus approach. *Logic Journal of the IGPL*, 22(5):737–757, October 2014. doi:10.1093/jigpal/jzu011.

62 Paulo A. S. Veloso, Sheila R. M. Veloso, and Mario R. F. Benevides. On a graph calculus for modalities. *Theoretical Computer Science*, 685:83–103, July 2017. doi:10.1016/j.tcs.2016.11.037.

63 Burghard Von Karger and C. A. R. Hoare. Sequential calculus. *Information Processing Letters*, 53(3):123–130, February 1995. doi:10.1016/0020-0190(94)00205-D.

64 Fernando Zalamea. Lógica topológica. una introducción a los gráficos existenciales de peirce. Memorias del XIV Coloquio Distrital de Matemáticas y Estadística, 1997.

65 Fernando Zalamea. Pragmaticismo, gráficos y continuidad. hacia el lugar de c. s. peirce en la historia de la lógica. Mathesis 13, pp. 147–156, 1997.

66 Fernando Zalamea. Peirce's logic of continuity: Existential graphs and non-Cantorian continuum. *Review of Modern Logic*, 9(1-2):115–162, January 2003. Publisher: The Review of Modern Logic. URL: https://projecteuclid.org/journals/review-of-modern-logic/volume-9/issue-1-2/Peirces-logic-of-continuity-Existential-graphs-and-non-Cantorian/rml/1081173838.full.

67 J.J. Zeman. *The Graphical Logic of C. S. Peirce*. PhD thesis, University of Chicago, 1964. URL: https://books.google.fr/books?id=E0AqAQAAMAAJ.

68 Bohua Zhan, Zhenyan Ji, Wenfan Zhou, Chaozhu Xiang, Jie Hou, and Wenhui Sun. Design of point-and-click user interfaces for proof assistants. In *Formal Methods and Software Engineering: 21st International Conference on Formal Engineering Methods, ICFEM 2019, Shenzhen, China, November 5–9, 2019, Proceedings*, pages 86–103, Berlin, Heidelberg, 2019. Springer-Verlag. doi:10.1007/978-3-030-32409-4_6.

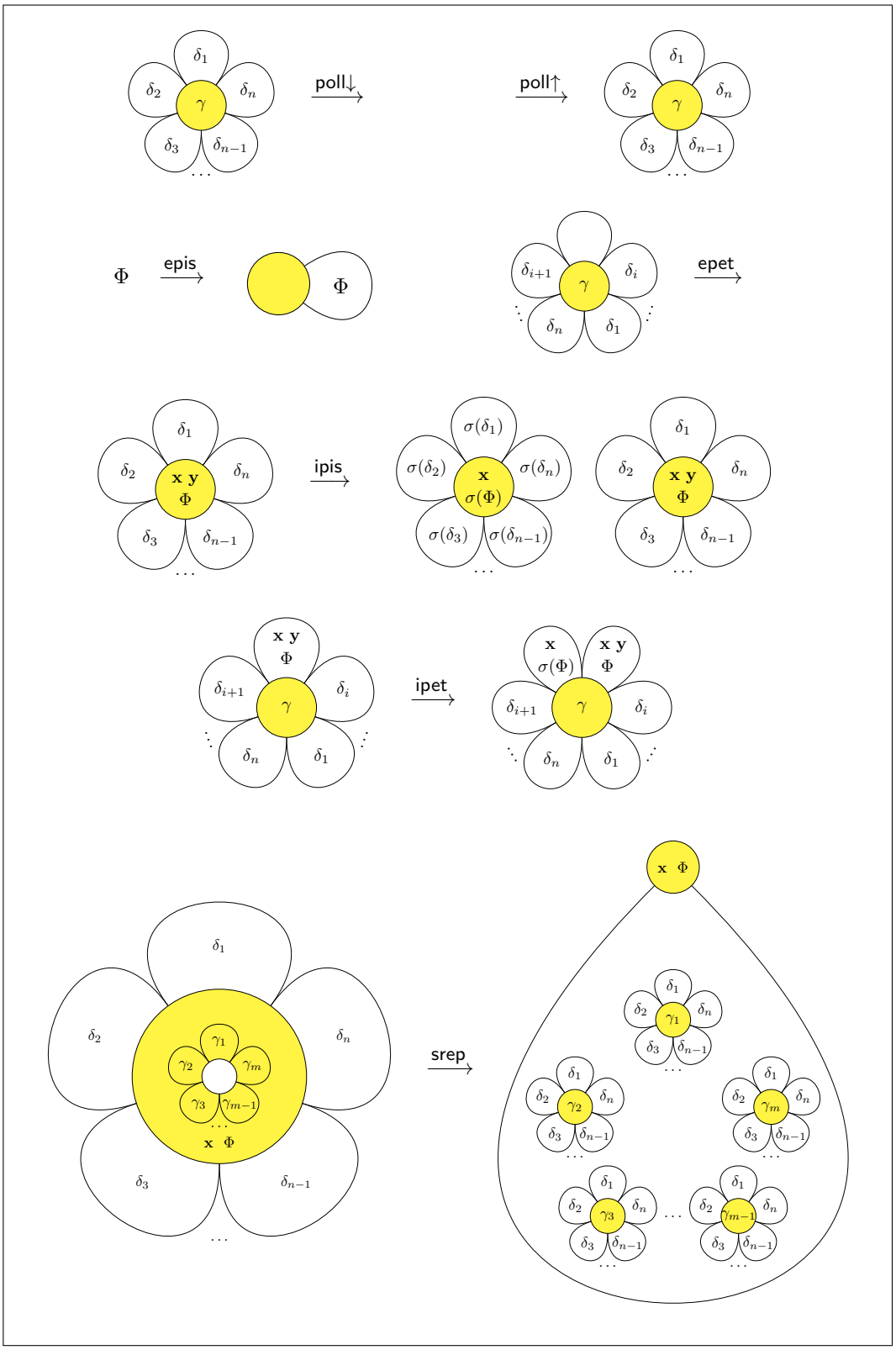


Figure 8 Graphical presentation of natural rules ❀.

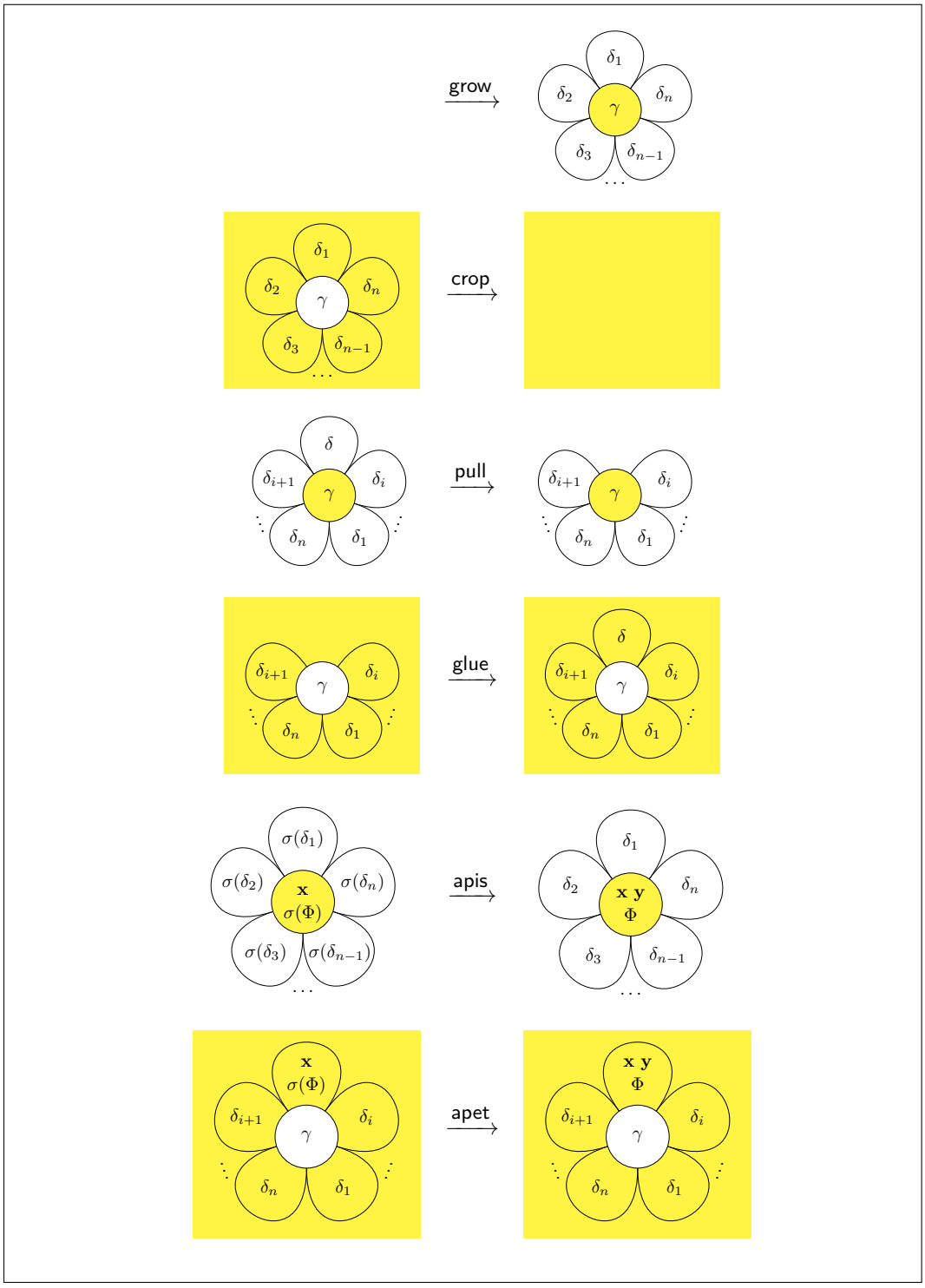
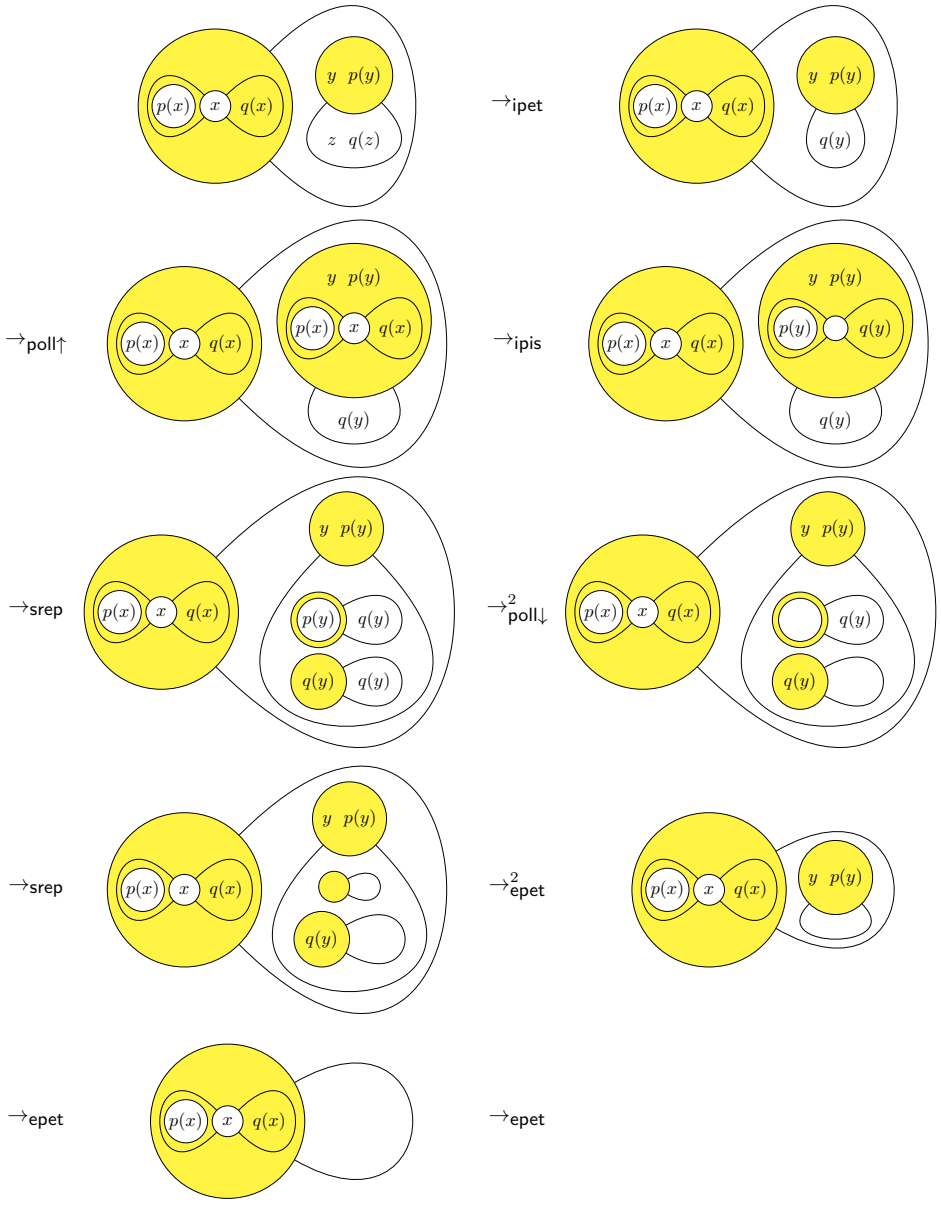


Figure 9 Graphical presentation of cultural rules ✂.

$$
\begin{array}{ll}
 & (x\cdot \mathsf{D}\ (p(x)\ \mathsf{D})\,;\, q(x))\ \mathsf{D}\ (y\cdot p(y)\ \mathsf{D}\ z\cdot q(z)) \\
\to_{\mathsf{ipet}} & (x\cdot \mathsf{D}\ (p(x)\ \mathsf{D})\,;\, q(x))\ \mathsf{D}\ (y\cdot p(y)\ \mathsf{D}\ q(y)) \\
\to_{\mathsf{poll}\uparrow} & (x\cdot \mathsf{D}\ (p(x)\ \mathsf{D})\,;\, q(x))\ \mathsf{D}\ (y\cdot p(y), (x\cdot \mathsf{D}\ (p(x)\ \mathsf{D})\,;\, q(x))\ \mathsf{D}\ q(y)) \\
\to_{\mathsf{ipis}} & (x\cdot \mathsf{D}\ (p(x)\ \mathsf{D})\,;\, q(x))\ \mathsf{D}\ (y\cdot p(y), (\mathsf{D}\ (p(y)\ \mathsf{D})\,;\, q(y))\ \mathsf{D}\ q(y)) \\
\to_{\mathsf{srep}} & (x\cdot \mathsf{D}\ (p(x)\ \mathsf{D})\,;\, q(x))\ \mathsf{D}\ (y\cdot p(y)\ \mathsf{D}\ ((p(y)\ \mathsf{D})\ \mathsf{D}\ q(y)), (q(y)\ \mathsf{D}\ q(y))) \\
\to^2_{\mathsf{poll}\downarrow} & (x\cdot \mathsf{D}\ (p(x)\ \mathsf{D})\,;\, q(x))\ \mathsf{D}\ (y\cdot p(y)\ \mathsf{D}\ ((\mathsf{D})\ \mathsf{D}\ q(y)), (q(y)\ \mathsf{D}\ \cdot)) \\
\to_{\mathsf{srep}} & (x\cdot \mathsf{D}\ (p(x)\ \mathsf{D})\,;\, q(x))\ \mathsf{D}\ (y\cdot p(y)\ \mathsf{D}\ (\mathsf{D}\ \cdot), (q(y)\ \mathsf{D}\ \cdot)) \\
\to^2_{\mathsf{epet}} & (x\cdot \mathsf{D}\ (p(x)\ \mathsf{D})\,;\, q(x))\ \mathsf{D}\ (y\cdot p(y)\ \mathsf{D}\ \cdot) \\
\to_{\mathsf{epet}} & (x\cdot \mathsf{D}\ (p(x)\ \mathsf{D})\,;\, q(x))\ \mathsf{D}\ \cdot \\
\to_{\mathsf{epet}} &
\end{array}
$$

(a) Textual presentation.



(b) Graphical presentation.

Figure 10 A natural proof in the flower calculus.

Delooping Generated Groups in Homotopy Type Theory

Camil Champin
École Normale Supérieure de Lyon, France

Samuel Mimram
LIX, CNRS, École polytechnique, Institut Polytechnique de Paris, Palaiseau, France

Émile Oleon
LIX, CNRS, École polytechnique, Institut Polytechnique de Paris, Palaiseau, France

Abstract

Homotopy type theory is a logical setting based on Martin-Löf type theory in which one can perform geometric constructions and proofs in a synthetic way. Namely, types can be interpreted as spaces (up to continuous deformation) and proofs as homotopy invariant constructions. In this context, loop spaces of pointed connected groupoids provide a natural representation of groups, and any group can be obtained as the loop space of such a type, which is then called a *delooping* of the group. There are two main methods to construct the delooping of an arbitrary group G. The first one consists in describing it as a pointed higher inductive type, whereas the second one consists in taking the connected component of the principal G-torsor in the type of sets equipped with an action of G. We show here that, when a presentation is known for the group, simpler variants of those constructions can be used to build deloopings. The resulting types are more amenable to computations and lead to simpler meta-theoretic reasoning. We also investigate, in this context, an abstract construction for the Cayley graph of a generated group and show that it encodes the relations of the group. Most of the developments performed in the article have been formalized using the cubical version of the Agda proof assistant.

2012 ACM Subject Classification Theory of computation → Constructive mathematics

Keywords and phrases homotopy type theory, delooping, group, generator, Cayley graph

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.6

Supplementary Material
Software (Proofs): `https://github.com/smimram/generated-deloopings-agda` [10]
archived at `swh:1:dir:4fc863802e40f99733703893622a7aa23c50c308`

Acknowledgements We would like to thank Dan Christensen as well as an anonymous reviewer for useful comments on early drafts of this article.

Introduction

Homotopy type theory was introduced around 2010 [27]. It is based on Martin-Löf type theory [20], starting from the idea that types in logic should be interpreted not only as sets, as traditionally done in semantics of logic, but rather as *spaces* considered up to homotopy. Namely, the identities between two elements of a type can be thought of as paths between points corresponding to the elements, identities on identities as homotopies between paths, and so on. Moreover, this correspondence can be made to work precisely, by postulating the *univalence axiom* [14], which states that identities between types coincide with equivalences. This opens the way to the implementation of geometric constructions in a synthetic way, by performing operations on types, which will semantically correspond to the desired operations on spaces. In this setting, we are interested in providing ways to construct models of groups which are concise in order to allow for simple proofs, but also to make the meta-theoretic reasoning easier.

Delooping groups. Following a well-known construction due to Poincaré at the end of the 19th century [22], to any type A which is pointed, i.e. equipped with a distinguished element $\star$, we can associate its *fundamental group* $\pi_1(A) := \|\star = \star\|_0$ whose elements are homotopy classes of paths from $\star$ to itself, with composition given by concatenation and identity by the constant path. Moreover, when the type A is a groupoid, in the sense that any two homotopies between paths are homotopic, this fundamental group coincides with the *loop space* $\Omega\, A := (\star = \star)$, defined similarly but without quotienting paths up to homotopy. Once this observation made, it is natural to wonder whether every group G arises as the loop space of some groupoid. It turns out that this is the case: to every group one can associate a pointed connected groupoid type $\mathrm{B}\, G$, called its *delooping*, whose loop space is G. Moreover, there is essentially only one such type, thus justifying the notation.

Internal and external points of view. The delooping construction, which can be found in various places [2, 5], and will be recalled in the article, induces an equivalence between the type of groups and the type of pointed connected groupoids (Theorem 20). This thus provides us with two alternative descriptions of groups in homotopy type theory. The one as (loop spaces of) pointed connected groupoids can be thought of as an *internal* one, since the structure is deduced from the types without imposing further axioms; by opposition, the traditional one as groups (sets equipped with multiplication and unit operations) is rather an *external* one (some also use the terminology *concrete* and *abstract* instead of internal and external [2]). We should also say here that pointed connected types (which are not necessarily groupoids) can be thought of as higher versions of groups, where the axioms only hold up to higher identities which are themselves coherent, and so on.

Two ways to construct deloopings. Two generic ways are currently known in order to construct the delooping $\mathrm{B}\, G$ of a group G, which we both refine in this article. The first one is a particular case of the definition of *Eilenberg-MacLane spaces* in homotopy type theory due to Finster and Licata [17]. It consists in constructing $\mathrm{B}\, G$ as a higher inductive type with one point (so that it is pointed), one loop for each element of G, one identity for each entry in the multiplication table of G, and then truncating the resulting type as a groupoid. One can imagine that the resulting space has the right loop space "by construction", although the formal proof is non-trivial.

The second one is the *torsor* construction which originates in algebraic topology [12] and can be adapted in homotopy type theory [2, 5, 29]. One can consider the type of *G-sets*, which are sets equipped with an action of G. Among those, there is a canonical one, called the *principal G-torsor* P_G, which arises from the action of the group G on itself by left multiplication. It can be shown that the loop space of the type of G-sets, pointed on the principal G-torsor P_G, is the group G. Moreover, if one restricts the type of G-sets to the connected component of the principal G-torsor, one obtains the type of *G-torsors*, which is a delooping of G.

Smaller deloopings of groups. In this article, we are interested in refining the above two constructions in order to provide ones which are "simpler" (in the sense that we have less constructors, or the definition requires to introduce less material), when a presentation by generators and relation is known for the group.

For the first construction (as a higher inductive type), we show here the we can construct $\mathrm{B}\, G$ as the higher inductive type generated by one point, one loop for each generator of the presentation (as opposed to every element of the group), one identity for each relation of

the presentation and taking the groupoid truncation (Theorem 2). This has the advantage of resulting in types are simpler to define, require handling less cases when reasoning with those by induction, and are closer to the usual combinatorial description of groups. Moreover, we claim that the traditional methods based on rewriting [13, 1] in order to compute invariants such as homology or coherence can be applied to those. Namely, a first important step in this direction was obtained by Kraus and von Raummer's adaptation of Squier's coherence theorem in homotopy type theory [16].

For the second construction (based on G-torsors), we show that a simpler definition can be achieved when a generating set X is known for G. Namely, we show that one can perform essentially the same construction, but replacing G-gets by what we call here X-sets (Theorem 11), where we only need to consider the action for the generators (as opposed to the whole group). As an illuminating example, consider the case $G = \mathbb{Z}$, whose delooping is known to be the circle $\mathrm{B}\,\mathbb{Z} = \mathrm{S}^1$. The type $\mathcal{U}^{\circlearrowleft}$ of all endomorphisms, on any type, contains, as a particular element, the successor function $s : \mathbb{Z} \to \mathbb{Z}$. Our results imply that the connected component of s in $\mathcal{U}^{\circlearrowleft}$ is a delooping of $\mathbb{Z}$. This description is arguably simpler than the one of $\mathbb{Z}$-torsors: indeed, morphisms of $\mathbb{Z}$-sets are required to preserve the action of every element of $\mathbb{Z}$, while morphisms in $\mathcal{U}^{\circlearrowleft}$ are only required to preserve the action of 1 (which corresponds to the successor). The above description is the one which is used in UniMath in order to define the circle S^1 [3]: the reason why they use it instead of more traditional one [27] is that they do not allow themselves to use higher inductive types because those are not entirely clear from a meta-theoretic point of view (there is no general definition, even though there are proposals [18], the semantics of type theory [14] has not been fully worked out in their presence, etc.). Our result thus give an abstract explanation about why this construction works and provides a generic way to easily define many more deloopings without resorting to higher inductive types, if one is not disposed to do so.

Cayley graphs. As a last aspect of our study of generated groups in homotopy type theory, we provide here a pleasant abstract description of Cayley graphs, which is a well-known construction in group theory [9, 19]. We show that, given a group G with a set X of generators, the Cayley graph can be obtained as the kernel of the canonical map $\mathrm{B}\,X^* \to \mathrm{B}\,G$, where X^* is the free group on X (Theorem 16). This establishes those graphs as a measure of the difference between deloopings and their approximations, and suggests higher dimensional versions of those.

Formalization. Most of the results presented in this article have been formalized in the cubical variant of the Agda proof assistant [28] using the "standard library" which has been developed for it [26]. Our developments are publicly available [10], and we provide pointers to the formalized results.

Plan of the paper. We begin by briefly recalling the fundamental notions of homotopy type theory which will be used throughout the paper (Section 1), as well as the notion of delooping for a group (Section 2). We first present the construction of deloopings using higher inductive types, and explain how those can be simplified when a presentation is known for the group (Section 3). We then present the other approach for defining delooping of groups based on the torsor construction (Section 4) and show how it can be simplified when a generating set is known for the group (Section 5). Finally, we investigate the construction of Cayley graphs in homotopy type theory (Section 6) and conclude, presenting possible extensions of this work (Section 7).

1 Homotopy type theory

We unfortunately do not have enough space here to provide an introduction from scratch to dependent type theory and homotopy type theory, so we refer the reader to the reference book for an in depth presentation [27]. The main purpose of this section is to fix some terminology and notations for classical notions.

Universe. We write $\mathcal{U}$ for the *universe*, i.e. the large type of all small types, which we suppose to be closed under dependent sums and products. We write $\Pi(x : A).B$ or $(x : A) \to B$ for Π-types, and $A \to B$ for the case where B is non-dependent. Similarly, we write $\Sigma(x : A).B$ for Σ-types, and $A \times B$ for the non-dependent version. The two projections from a Σ-type are respectively written π and π'.

Paths. Given a type $A : \mathcal{U}$ and two elements $a, b : A$, we write $a =_A b$ for the type of *identities*, or *paths*, between a and b: its elements are proofs of equality between a and b. In particular, for any $a : A$, the type $a = a$ contains the term refl_a witnessing for reflexivity of equality. We sometimes write $x := t$ to indicate that x and t are equal by definition. The elimination principle of identities, aka *path induction* and often noted J, roughly states that, given $a : A$, in order to show a property $P : (x : A) \to (a = x) \to \mathcal{U}$ for every $x : A$ and $p : a = x$ it is enough to show it in the case where $x := a$ and $p := \mathrm{refl}_a$. By path induction, the following can be shown. Given a type A and a type family $B : A \to \mathcal{U}$, a path $p : x = y$ in A induces a function $B_p^{\to} : B(x) \to B(y)$ witnessing for the fact that equality is *substitutive*. As a special case, any path $p : A = B$ between two types $A, B : \mathcal{U}$ induces a function $p^{\to} : A \to B$, called the *transport* along p, as well as an inverse function $p^{\leftarrow} : B \to A$. Finally, given a function $f : A \to B$, any path $p : x = y$ in A induces a path $f^{=}(p) : f(x) = f(y)$ witnessing for the fact that equality is a *congruence*.

Higher inductive types. Many functional programming languages allow the definition of inductive types, which are freely generated by constructors. For instance, the type S^0 of booleans is generated by two elements (true and false). In the context of homotopy type theory, languages such as cubical Agda feature a useful generalization of such types, called *higher inductive types*. They allow, in addition to traditional constructors for elements of the type, constructors for equalities between elements of the type. For instance, the type corresponding to the circle S^1 can be defined as generated by two points a and b and two equalities $p, q : a = b$ between those points. Higher-dimensional spheres S^n can be defined in a similar way.

Univalence. A map $f : A \to B$ is an *equivalence* when it admits both a left and a right inverse. In particular, every isomorphism is an equivalence. We write $A \simeq B$ for the type of equivalences from A to B. The identity is clearly an equivalence and we thus have, by path induction, a canonical map $(A = B) \to (A \simeq B)$ for every types A and B: the *univalence axiom* states that this map is itself an equivalence. In particular, every equivalence $A \simeq B$ induces a path $A = B$. It is known that univalence implies the *function extensionality* principle [27, Section 2.9]: given functions $f, g : A \to B$, if $f(x) = g(x)$ for any $x : A$ then $f = g$ (and the expected generalization to dependent function types is also valid).

Homotopy levels. A type A is *contractible* when the type $\Sigma(x : A).(y : A) \to (x = y)$ is inhabited: this means that we have a "contraction point" $a_0 : A$, and a continuous family of paths from a_0 to every other point in A. A type A is a *proposition* (resp. a *set*, resp. a

groupoid) when $(x = y)$ is contractible (resp. a proposition, resp. a set) for every $x, y : A$. Intuitively, a contractible type is a point (up to homotopy), a proposition is a point or is empty, a set is a collection of points and a groupoid is a space which bears no non-trivial 2-dimensional (or higher) structure. We write Set for the type of sets. Given a type A, we write isSet(A) (resp. isGroupoid(A)) for the predicate indicating that A is a set (resp. groupoid).

Truncation. Given a type A, its *propositional truncation* turns it into a proposition in a universal way. It consists of a type $\|A\|_{-1}$, which is a proposition, equipped with a map $|-|_{-1} : A \to \|A\|_{-1}$ such that, for any proposition B, the map $(\|A\|_{-1} \to B) \to (A \to B)$ induced by precomposition by $|-|_{-1}$ is an equivalence. Intuitively, the type $\|A\|_{-1}$ behaves like A, except that we do not have access to its individual elements: the elimination principle for propositional truncation states that in order to construct an element of B from an element of $\|A\|_{-1}$, we can only assume that we have an element of A if B itself is a proposition. The *set truncation* $\|A\|_0$ of a type A is defined similarly, as the universal way of turning A into a set, and we write $|x|_0$ for the image of $x : A$ in the truncation; and we can similarly define the *groupoid truncation* $\|A\|_1$.

Fibers. Given a function $f : A \to B$, we write $\mathrm{fib}_f\, b$ for the type $\Sigma(a : A).(f\, a = b)$, called the *fiber* of f at b. The function f is said to be *surjective* when the type $(b : B) \to \|\,\mathrm{fib}_f\, b\|_{-1}$ is inhabited, i.e. when every element of B merely admits a preimage.

2 Delooping groups

The external point of view. A *group* consists of a set A, together with an operation $m : A \to A \to A$ (the *multiplication*), an element $e : A$ (the *unit*), and an operation $i : A \to A$ (the *inverse*) such that multiplication is associative, admits e as unit, and $i(x)$ is the two-sided inverse of any element $x : A$. We write Group for the type of all groups, and $G \to_{\mathrm{Grp}} H$ for the type of group morphisms between groups G and H. In the following, we use the traditional notations for groups: given two elements $x, y : G$, we simply write xy instead of $m(x, y)$, 1 instead of e, and x^{-1} instead of $i(x)$.

The internal point of view. A *pointed* type consists of a type A together with a distinguished element, often written $\star$ and sometimes left implicit. Given a pointed type $(A, \star)$, its *loop space* $\Omega\, A$ is defined as the type of paths from $\star$ to itself: $\Omega\, A := (\star = \star)$. The elements of this type are called *loops*. By path induction one can construct, for every two paths $p : a = b$ and $q : b = c$, a path in $a = c$ called their *concatenation* and written $p \cdot q$. Similarly, every path $p : a = b$, admits an *inverse* path $p^- : b = a$. When A is a pointed groupoid, $\Omega\, A$ is a set, and these operations canonically equip this set with a structure of group [27, Section 2.1].

Delooping groups. A *delooping* of a group G is a pointed connected groupoid $\mathrm{B}\, G$ together with an identification $\mathrm{d}_G : \Omega\, \mathrm{B}\, G = G$ (we recall that a type A is *connected* when the type $\|A\|_0$ is contractible, i.e. A has one connected component). The notation is justified by the fact that deloopings are unique. For instance, it is known that the circle is a delooping of $\mathbb{Z}$: indeed, S^1 is a connected groupoid, and its fundamental group is $\mathbb{Z}$ [27, Section 8.1].

3 Delooping using higher inductive types

Delooping as a higher inductive type. Given a group G, its delooping should have a point $\star$ and a loop for every element of the group. Moreover, we should ensure that the multiplication of G coincides with the concatenation operation on the loop space, and that the type we obtain is a (pointed connected) groupoid. This suggests considering a higher inductive type, noted $\mathrm{K}(G,1)$, with the following constructors

$$\begin{array}{ll} \star & : \mathrm{K}(G,1) \\ \mathsf{loop} & : G \to \star = \star \\ \mathsf{loop\text{-}comp} & : (x,y:G) \to \mathsf{loop}\,x \cdot \mathsf{loop}\,y = \mathsf{loop}(xy) \\ \mathsf{trunc} & : \mathrm{isGroupoid}(\mathrm{K}(G,1)) \end{array}$$

This construction was first proposed by Finster and Licata. They also showed, using the encode-decode method, that it is a delooping of the original group, i.e. $\Omega\,\mathrm{K}(G,1) = G$, see [17, Theorem 3.2]. Note that we only ask here that loop preserves multiplication (with loop-comp), because it can be shown that this implies preservation of unit and inverses. In particular, preservation of unit (see `EM.loop-id`) renders superfluous one of the constructors present in the original definition [17].

In the following, we will define a variant of this higher inductive type when the group G is presented, which is smaller and gives rise to computations closer to traditional group theory.

Presentations of groups. Given a set X, we write X^* for the *free group* over X [27, Theorem 6.11.6]. There is an inclusion function $\iota : X \to X^*$ which, by precomposition, induces an equivalence between morphisms of groups $X^* \to G$ and functions $X \to G$. We write $f^* : X^* \to G$ for the group morphism thus induced by a function $f : X \to G$. The elements of X^* can be described as formal composites $a_1 \dots a_n$ where each a_i is an element of X or a formal inverse of an element of X (such that an element with an adjacent formal inverse cancel out).

Any free group X^* admits a delooping as a wedge of an X-indexed familly of circles. The corresponding type $\bigvee_X \mathrm{S}^1$ can be described as the coequalizer

$$X \rightrightarrows 1 \dashrightarrow \bigvee_X \mathrm{S}^1 \tag{1}$$

or, equivalently, as the higher inductive type generated by the two constructors $\star : \bigvee_X \mathrm{S}^1$ and $\mathsf{loop} : X \to \star = \star$.

▶ **Proposition 1.** *We have $\Omega \bigvee_X \mathrm{S}^1 = X^*$, i.e. the above type is a $\mathrm{B}\,X^*$.*

Proof. The fact that $\bigvee_X \mathrm{S}^1$ is a delooping of X^* is not too difficult to show when X has decidable equality, see [27, Exercise 8.2] and [15], but the general case is more involved and was recently proved in [30]: the main issue is to show that this type is a groupoid. ◀

A group *presentation* $\langle X \mid R \rangle$ consists of a set X of *generators*, a set R of *relations*, and two functions $\pi, \pi' : R \to X^*$ respectively associating to a relation its *source* and *target*. We often write $r : u \Rightarrow v$ for a relation r with u as source and v as target. Given such a presentation P, the corresponding *presented* group $[P]$ is the set quotient X^*/R of the free group on X under the smallest congruence identifying the source and the target of every relation $r : R$. This type can be described as the type $[P] := \|X^* /\!\!/ R\|_0$ obtained by taking the set truncation of the coequalizer

$$R \underset{\pi'}{\overset{\pi}{\rightrightarrows}} X^* \overset{\kappa}{\dashrightarrow} X^* /\!\!/ R$$

From this also follows a description of $[P]$ as a higher inductive type:

$$\begin{aligned}&\mathsf{word} : X^* \to [P]\\ &\mathsf{rel} \quad : (r : R) \to \mathsf{word}(\pi(r)) = \mathsf{word}(\pi'(r))\\ &\mathsf{trunc} : \mathrm{isSet}([P])\end{aligned}$$

A smaller delooping. Suppose given a group G along with a presentation $P := \langle X \mid R\rangle$, i.e. such that $G = [P]$. We define the type $\mathrm{B}\,P$ as the following higher inductive type:

$$\begin{aligned}&\star \quad : \mathrm{B}\,P\\ &\mathsf{gen} \quad : X \to (\star = \star)\\ &\mathsf{rel} \quad : (r : R) \to (\mathsf{gen}^*(\pi(r)) = \mathsf{gen}^*(\pi'(r)))\\ &\mathsf{trunc} : \mathrm{isGroupoid}(\mathrm{B}\,P)\end{aligned}$$

This type is generated by a point $\star$, then the constructor gen adds a loop $\underline{a} : \star = \star$ for every generator a, the constructor rel adds an equality $\underline{a}_1 \cdot \underline{a}_2 \cdot \ldots \cdot \underline{a}_n = \underline{b}_1 \cdot \underline{b}_2 \cdot \ldots \cdot \underline{b}_m$ for each relation $a_1 \ldots a_n \Rightarrow b_1 \ldots b_m$, and the constructor trunc formally takes the groupoid truncation of the resulting type. Note that, because of the presence of $\mathsf{gen}^* : X^* \to (\star = \star)$ in the type of rel, the above inductive type is not accepted as is in standard proof assistants such as Agda. However, a definition can be done in two stages, by first considering $\bigvee_X \mathrm{S}^1$ (i.e. the type generated only by $\star$ and gen), and then defining a second inductive type further quotienting this type (i.e. adding the constructors rel and trunc), see `EM.Delooping`. Also, the definition of gen^* requires the group structure on $\bigvee_X \mathrm{S}^1$: the group operations are easily defined from operations on paths (reflexivity, concatenation, symmetry), but the fact that it is a groupoid is non-trivial (see Proposition 1). Our main result in this section is the following:

▶ **Theorem 2** (`EM.theorem`). *Given a presentation $P := \langle X \mid R\rangle$, the type $\mathrm{B}\,P$ is a delooping of the group $[P]$.*

Proof. By induction on $\mathrm{B}\,P$, we can define a function $f : \mathrm{B}\,P \to \mathrm{K}([P], 1)$ such that $f\star := \star$, and $f(\mathsf{gen}\,a) := \mathsf{loop}[a]$ for all $a : X$. It can be shown that f is then such that $f^=(\mathsf{gen}^*\,u) = \mathsf{loop}[u]$, for any $u : X^*$. We can therefore define the image $f^=(r)$ on a relation $r : u \Rightarrow v$ as the composite of equalities

$$f^=(\mathsf{gen}^*\,u) = \mathsf{loop}[u] = \mathsf{loop}[v] = f^=(\mathsf{gen}^*\,v)$$

where the equality in the middle follows from the fact that we have $[u] = [v]$ because of the relation r.

In the other direction, the group morphism $\mathsf{gen}^* : X^* \to \Omega\,\mathrm{B}\,P$ preserves relations (by rel), and thus induces a quotient morphism $g' : [P] \to \Omega\,\mathrm{B}\,P$. We can thus consider the function $g : \mathrm{K}([P], 1) \to \mathrm{B}\,P$ such that $g(\star) = \star$, for $x : [P]$ we have $g^=(\mathsf{loop}\,x) = g'(x)$, and for $x, y : [P]$ the image of $\mathsf{loop\text{-}comp}\,x\,y$ is canonically induced by the fact that g' preserves group multiplication.

Since $\mathrm{K}([P], 1)$ is a groupoid, in order to show that $f(g(x)) = x$ for every $x : \mathrm{K}([P], 1)$, it is enough to show that it holds for $x := \star$, which is the case by definition of f and g, and that this property is preserved under $\mathsf{loop}\,x$ for $x : [P]$, which follows from the fact that we have $f^=(g'(x)) = \mathsf{loop}\,x$ for any $x : [P]$ (this is easily shown by induction on x). Conversely, we have to show that $g(f(x)) = x$ holds for $x : \mathrm{B}\,P$. Again, this is shown by induction on x. ◀

As an interesting remark, the careful reader will note that the fact that the types X and R are sets does not play a role in the proof: in fact, those assumptions can be dropped here. Also, note that we do not need the choice of a representative in X^* for every element of $[P]$

in order to define the function g from gen^* in the above proof: intuitively, this is because the induced function g does not depend on such a choice of representatives. Finally, we should mention here that a similar result is mentioned as an exercise in [27, Example 8.7.17]; the proof suggested there is more involved since it is based on a generalized van Kampen theorem.

▶ **Example 3.** The dihedral group D_5, see Example 13, admits the presentation

$$\langle r, s \mid r^4 = srs, sr^2s = r^3, rsr = s, r^3s = sr^2, sr^3 = r^2s, s^2 = 1\rangle$$

Hence, by Theorem 2 we can construct a delooping of D_5 as an higher inductive type generated by two loops (corresponding to r and s) and six 2-dimensional cells (corresponding to the relations). Note that this is much smaller than $\mathrm{K}(D_5, 1)$ (it has 2 instead of 10 generating loops, and 6 instead of 100 relations), thus resulting in shorter proofs when reasoning by induction.

▶ **Example 4.** Any group G admits a presentation, the *standard presentation*, with one generator $\underline{a}$ for every element $a : G$, and relations $\underline{a}\ \underline{b} = \underline{ab}$ for every pair of generators, as well as $\underline{1} = 1$. By applying Theorem 2, we actually recover the inductive type $\mathrm{K}(G, 1)$ as delooping of G.

4 Delooping with torsors

In this section, we recall the other classical approach to constructing deloopings of groups by using G-torsors, which originates in classical constructions of algebraic topology [12]. Most of the material of the section is already known, for which reason proofs are not much detailed. A more in-depth presentation can be found in recent works such as [2, 5].

Group actions. Given a group G and a set A, an *action* of G on A is a group morphism from G to $A \simeq A$, that is a map $\alpha : G \to (A \simeq A)$ such that

$$\alpha(xy) = \alpha(x) \circ \alpha(y) \qquad \alpha(1) = \mathrm{id}_A \tag{2}$$

for all $x, y : G$.

A *G-set* is a set equipped with an action of G, and we write Set_G for the type of G-sets. We often simply denote a G-set by the associated action α and write $\mathrm{dom}(\alpha)$ for the set on which G acts.

▶ **Lemma 5** (`GSetProperties.isGroupoidGSet`). *The type Set_G is a groupoid.*

Proof. The type of sets is a groupoid [27, Theorem 7.1.11]. Given a set A, the type of functions $A \to A$ is a set [27, Theorem 7.1.9] and thus a groupoid. Finally, the axioms (2) of actions are propositions (because A is a set) and thus groupoids. We conclude since groupoids are closed under Σ-types [27, Theorem 7.1.8]. ◀

Given G-sets α and β, a *morphism* between them consists of a function $f : \mathrm{dom}\,\alpha \to \mathrm{dom}\,\beta$ which preserves the group action, in the sense that for every $x : G$ and $a : \mathrm{dom}\,\alpha$, we have

$$\beta(x)(f(a)) = f(\alpha(x)(a)). \tag{3}$$

A morphism which is also an equivalence is called an *isomorphism* and we write $\alpha \simeq^{\mathrm{Set}_G} \beta$ for the type of isomorphisms between α and β. We write $\mathrm{Aut}(\alpha)$ for the type of automorphisms $\alpha \simeq^{\mathrm{Set}_G} \alpha$, which is a group under composition. The equalities between G-sets can be conveniently characterized as follows.

▶ **Proposition 6** (`GSetProperties.GSet≡Decomp`). *Given two G-sets α and β, an equality between them consists of an equality $p : \operatorname{dom}\alpha = \operatorname{dom}\beta$ such that the function induced by transport along p, namely $p^{\rightarrow} : \operatorname{dom}\alpha \to \operatorname{dom}\beta$, is a morphism of G-sets.*

Proof. The characterization of equalities between Σ-types [27, Theorem 2.7.2] entails that an equality between $(\operatorname{dom}\alpha, \alpha)$ and $(\operatorname{dom}\beta, \beta)$ is a pair consisting of an equality $p : \operatorname{dom}\alpha = \operatorname{dom}\beta$ and an equality $q : p^{\rightarrow}(\alpha) = \beta$ (we can forget about the equality between the components expressing the properties required for group actions since those are propositions). By [27, Lemma 2.9.6] and function extensionality, we finally have that the type of q is equivalent to the type $\beta(x) \circ p^{\rightarrow} = p^{\rightarrow} \circ \alpha(x)$. ◀

It easily follows from this proposition that any equality between G-sets induces an isomorphism of G-sets, as customary for equalities between algebraic structures [27, Section 2.14]. In fact, this map from equalities to isomorphisms can itself be shown to be an equivalence:

▶ **Proposition 7** (`GSetProperties.GSetPath`). *Given G-sets α and β, the canonical function*

$$(\alpha = \beta) \to (\alpha \simeq^{\mathrm{Set}_G} \beta)$$

is an equivalence. Moreover, given a G-set α, the induced equivalence

$$(\alpha = \alpha) \simeq (\alpha \simeq^{\mathrm{Set}_G} \alpha)$$

is compatible with the canonical group structures on both types.

Proof. This is actually an instance of a more general correspondence between equalities and isomorphisms of algebraic structures, which is known under the name of *structure identity principle*, see [11] and [27, Section 9.8], and can be understood as a generalisation of univalence for types having an algebraic structure. ◀

Torsors. For any group G, there is a canonical G-set called the *principal G-torsor* and noted P_G, corresponding to the action of G on itself by left multiplication. Moreover, its group of automorphisms is precisely the group G:

▶ **Proposition 8** (`Deloopings.PGloops`). *Given a group G, we have an equality of groups*

$$(\mathrm{P}_G \simeq^{\mathrm{Set}_G} \mathrm{P}_G) = G$$

Proof. The two functions $\phi : \operatorname{Aut}\mathrm{P}_G \to G$ and $\psi : G \to \operatorname{Aut}\mathrm{P}_G$, respectively defined by $\phi(f) := f(1)$ and $\psi(x)(y) := yx$ are mutually inverse group morphisms, see Appendix A. ◀

The type Set_G is thus "almost" a delooping of G. Namely, it is a groupoid (Lemma 5), which is pointed by P_G and satisfies $\Omega\,\mathrm{Set}_G = G$ by Propositions 7 and 8. It only lacks being connected, which can easily be addressed. Given a pointed type A, its *connected component* $\operatorname{Comp} A$ is the type $\Sigma(x : A).\|\star = x\|_{-1}$. It is well-known that this type is pointed by $(\star, |\operatorname{refl}|_{-1})$, connected and has the same loop space as the original type, i.e. we have $\Omega \operatorname{Comp} A = \Omega A$, see Appendix C. We thus have:

▶ **Theorem 9** (`Deloopings.torsorDeloops`). *The connected component of P_G in Set_G, i.e. the type $\operatorname{Comp}(\mathrm{Set}_G, \mathrm{P}_G)$, is a delooping of G.*

The elements of the connected component of the principal G-set are usually called *G-torsors*.

5 Generated torsors

Fix a group G. Given a set X and a map $\gamma : X \to G$, we say that X *generates* G (with respect to γ) when $\gamma^* : X^* \to G$ is surjective. From now on, we suppose that we are in such a situation. We now provide a variant for the construction of a delooping of G by G-torsors described in the previous section, taking advantage of the additional data of a generating set in order to obtain smaller and simpler constructions. Note that here, contrarily to Section 3, we only need a set of generators, not a full presentation.

Actions of sets. Given a type A, we write $\operatorname{End} A$ for its type of *endomorphisms*, i.e. maps $A \to A$. An *action* of the set X on a set A is a morphism $X \to \operatorname{End} A$, i.e. a family of endomorphisms of A indexed by X. We write Set_X for the type

$$\operatorname{Set}_X \quad := \quad \Sigma(A : \operatorname{Set}).(X \to \operatorname{End} A)$$

of actions of X. An element α of this type consists in a set $\operatorname{dom}\alpha$ with a function $\alpha : X \to \operatorname{End}(\operatorname{dom}\alpha)$ and is called an *X-set*. A *morphism* between X-sets α and β is a function $f : \operatorname{dom}\alpha \to \operatorname{dom}\beta$ satisfying (3) for every $x : X$. The identities between X-sets can be characterized in a similar way as for G-sets, see Proposition 6, and Proposition 7 also extends in the expected way.

Precomposition by γ induces a function $U : \operatorname{Set}_G \to \operatorname{Set}_X$ which can be thought of as a forgetful functor from G-sets to X-sets. Note that U depends on γ but we leave it implicit for concision.

Applications of the generated delooping. We have seen in the previous section that the connected component of the principal G-torsor P_G in G-sets is a delooping of G. Our aim in this section is to show here that this construction can be simplified by taking the connected component of the restriction of P_G to X-sets.

Before proving this theorem, which is formally stated as Theorem 11 below, we shall first illustrate its use on a concrete example. Consider $\mathbb{Z}_n$, the cyclic group with n elements. We write $s : \mathbb{Z}_n \to \mathbb{Z}_n$ for the successor (modulo n) function, which is an isomorphism. By Theorem 9, we know that the type

$$\Sigma(A : \operatorname{Set}_{\mathbb{Z}_n}).\| \operatorname{P}_{\mathbb{Z}_n} = A\|_{-1}$$

of $\mathbb{Z}_n$-torsors is a model of $\operatorname{B}\mathbb{Z}_n$. This type is the connected component of the principal $\mathbb{Z}_n$-torsor $\operatorname{P}_{\mathbb{Z}_n}$ in the universe $\operatorname{Set}_{\mathbb{Z}_n}$ of sets with an action of $\mathbb{Z}_n$, i.e. sets A equipped with a morphism $\alpha : \mathbb{Z}_n \to \operatorname{Aut} A$. Such a set A is thus comes with one automorphism $\alpha(k)$ for every element $k : \mathbb{Z}_n$, therefore k automorphisms in this case. However, most of them are superfluous: 1 generates all the elements of $\mathbb{Z}_n$ by addition, so $\alpha(1)$ generates all the $\alpha(k)$ by composition because $\alpha(k) = \alpha(1)^k$. The useful data of a $\mathbb{Z}_n$-set thus boils down to a set A together with one automorphism $\alpha : \operatorname{Aut} A$ such that $\alpha^n = \operatorname{id}_A$.

Indeed, writing $\operatorname{Set}^{\circlearrowleft} := \Sigma(A : \operatorname{Set}).\operatorname{End} A$ for the type of all *endomorphisms* (on any set), our theorem will imply that the type

$$\Sigma((A, f) : \operatorname{Set}^{\circlearrowleft}).\|(\mathbb{Z}_n, s) = (A, f)\|_{-1} \tag{4}$$

(the connected component of the successor modulo n in the universe of set endomorphisms) is still a delooping of $\mathbb{Z}_n$. Note that we didn't assume that f is an isomorphism nor that it should verify $f^n = \mathrm{id}$. This is because both properties follow from the fact that f is in the connected component of the successor (which satisfies those properties). Similarly, we do not need to explicitly assume that the domain of the endomorphism is a set.

Our theorem thus allows to define, in a relatively simple way, types corresponding to deloopings of groups. As recalled in the introduction, this is particularly useful when one is not disposed to use higher inductive types (e.g. because their definition, implementation and semantics are not entirely mature). This is in fact the reason why this approach was used in UnitMath to define the circle [3], and we provide a generic way to similarly define other types. We expect that it can be used to reason about groups and compute invariants such as their cohomology [8, 6, 4]. On a side note, one might be worried by the fact that we are "biased" (by using a particular set of generators), which allows us to be more concise but might make more difficult generic proofs compared to G-torsors: we expect that this is not the case because in order to define the group G itself, one usually needs to resort to a presentation, and thus is also biased in some sense...

The generated delooping. In the following, we write P_X for $U\,\mathrm{P}_G$.

▶ **Proposition 10** (`XSetProperties.theorem`). *We have a group equivalence* $\Omega\,\mathrm{P}_G \simeq \Omega\,\mathrm{P}_X$.

Proof. From Proposition 6, an element of $\Omega\,P_G$ consists of an equality $p : G = G$ in $\mathcal{U}$ such that

$$\mathrm{P}_G(x) \circ p^{\rightarrow} = p^{\rightarrow} \circ \mathrm{P}_G(x)$$

for every $x : G$. By function extensionality and the definition of the action P_G, this is equivalent to requiring, for every $g, z : G$ that

$$g(p^{\rightarrow}(z)) = p^{\rightarrow}(gz) \tag{5}$$

Note that the above equality is between elements of G, which is a set, and is thus a proposition. Similarly, an element of $\Omega\,\mathrm{P}_X$ consists of an equality $p : G = G$ in $\mathcal{U}$ satisfying

$$\gamma(x)(p^{\rightarrow}(z)) = p^{\rightarrow}(\gamma(x)z) \tag{6}$$

for every $x : X$ and $z : G$.

Clearly, any equality $p : G = G$ in $\Omega\,\mathrm{P}_G$ also belongs to $\Omega\,\mathrm{P}_X$ since the condition (6) is a particular case of (5). We thus have a function $\phi : \Omega\,\mathrm{P}_G \to \Omega\,\mathrm{P}_X$. Conversely, consider an element $p : G = G$ of $\Omega\,\mathrm{P}_X$, i.e. satisfying (6) for every $x : X$ and $z : G$. Our aim is to show that it belongs to $\Omega\,\mathrm{P}_G$. Given $g, z : G$, we thus want to show that (5) holds. Since γ^* is surjective, because X generates G, we know that there merely exists an element u of X^* such that $\gamma^*(u) = x$. Since (5) is a proposition, by the elimination principle of propositional truncation, we can actually suppose given such a u, and we have

$$\begin{aligned}
x(p^{\rightarrow}(y)) &= \gamma^*(u)(p^{\rightarrow}(y)) && \text{since } \gamma^*(u) = x \\
&= p^{\rightarrow}(\gamma^*(u)y) && \text{by repeated application of (6)} \\
&= p^{\rightarrow}(xy) && \text{since } \gamma^*(u) = x.
\end{aligned}$$

The second equality essentially corresponds to the commutation of the following diagram, where $u := x_1 x_2 \dots x_n$ with $x_i : X$:

$$\begin{array}{ccccccccc}
G & \xrightarrow{\gamma(x_1)} & G & \xrightarrow{\gamma(x_2)} & G & \longrightarrow \dots \longrightarrow & G & \xrightarrow{\gamma(x_n)} & G \\
{\scriptstyle p^{\rightarrow}}\downarrow & & {\scriptstyle p^{\rightarrow}}\downarrow & & {\scriptstyle p^{\rightarrow}}\downarrow & & \downarrow{\scriptstyle p^{\rightarrow}} & & \downarrow{\scriptstyle p^{\rightarrow}} \\
G & \xrightarrow[\gamma(x_1)]{} & G & \xrightarrow[\gamma(x_2)]{} & G & \longrightarrow \dots \longrightarrow & G & \xrightarrow[\gamma(x_n)]{} & G
\end{array}$$

This thus induces a function $\psi : \Omega\,\mathrm{P}_X \to \Omega\,\mathrm{P}_G$. The functions ϕ and ψ clearly preserve the group structure (given by concatenation of paths) and are mutually inverse of each other, hence we have the equivalence we wanted. ◀

▶ **Theorem 11.** *The type* $\mathrm{Comp}\,\mathrm{P}_X$ *is a delooping of* G*.*

Proof. Since taking the connected component preserves loops spaces (Proposition 23), we have that $\Omega\,\mathrm{Comp}\,\mathrm{P}_X$ is equal to $\Omega\,\mathrm{P}_X$, which in turn is equal to $\Omega\,\mathrm{P}_G$ by Proposition 10, and thus to G by Theorem 9. ◀

The delooping of G constructed in previous theorem is the component of P_X in X-sets:

$$\Sigma(A : \mathcal{U}).\Sigma(S : \mathrm{isSet}\,A).\Sigma(f : X \to \mathrm{End}\,A).\|\,\mathrm{P}_X = (A, S, f)\|_{-1}$$

Since for any type A, the type $\mathrm{isSet}\,A$ is a proposition [27, Theorem 7.1.10], and the underlying type of P_X is a set, the underlying type of any X-set in the connected component of P_X will also be a set. As a consequence, the above type can slightly be simplified, by dropping the requirement that A should be a set:

▶ **Proposition 12.** *The type* $\Sigma(A : \mathcal{U}).\Sigma(f : X \to \mathrm{End}\,A).\|\,\mathrm{P}_X = (A, f)\|_{-1}$ *is a delooping of* G*.*

For instance, the delooping (4) of $\mathbb{Z}_n$ can slightly be simplified as

$$\Sigma((A, f) : \mathcal{U}^{\circlearrowleft}).\|(\mathbb{Z}_n, s) = (A, f)\|_{-1}$$

where $\mathcal{U}^{\circlearrowleft} := \Sigma(A : \mathcal{U}).\,\mathrm{End}\,A$ is the type of all endomorphisms of the universe.

▶ **Example 13.** Theorem 11 applies to every group for which a generating set is known (and, of course, the smaller the better). For instance, given a natural number n, the *dihedral group* D_n is the group of symmetries of a regular polygon with n sides. It has $2n$ elements and is generated by two elements s (axial symmetry) and r (rotation by an angle of $2\pi/n$). Hence the connected component of the symmetry and the rotation in the type of pairs of set endomorphisms, i.e.

$$\Sigma(A : \mathrm{Set}).\Sigma(f, g : \mathrm{End}\,A \times \mathrm{End}\,A).\|(D_n, s, r) = (A, f, g)\|_{-1}$$

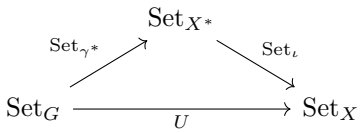
is a delooping of the dihedral group D_n.

Alternative proof. We would like to provide another proof Proposition 10, which was suggested by an anonymous reviewer. It is based on the idea that in order to show $\Omega\,\mathrm{P}_X \simeq \Omega\,\mathrm{P}_G$, it is enough to show that $U : \mathrm{Set}_G \to \mathrm{Set}_X$ is an *embedding*, i.e. that for every $\alpha, \beta : \mathrm{Set}_G$ the induced function $U^{=} : (\alpha = \beta) \to (U\alpha = U\beta)$ between path spaces is an equivalence [27, Definition 4.6.1]. It relies on the following result whose proof can be found in Appendix A.

▶ **Lemma 14.** *Given a type* $A : \mathcal{U}$*, type families* $P, Q : A \to \mathcal{U}$ *and* $f : (a : A) \to P\,a \to Q\,a$*, the map* $\Sigma A.f : \Sigma A.P \to \Sigma A.Q$ *is an embedding if and only if* $f\,a : P\,a \to Q\,a$ *is an embedding for every* $a : A$*.*

▶ **Proposition 15.** *The map* $U : \mathrm{Set}_G \to \mathrm{Set}_X$ *is an embedding.*

Proof. Given a morphism of groups $f : H \to K$, we write $\mathrm{Set}_f : \mathrm{Set}_K \to \mathrm{Set}_H$ for the function induced by precomposition. In particular, by definition, we have $U := \mathrm{Set}_\gamma$. The function $\gamma : X \to G$ can be decomposed as $\gamma^* \circ \iota$, and therefore Set_γ can be decomposed as $\mathrm{Set}_\iota \circ \mathrm{Set}_{\gamma^*}$:



Both maps are embeddings so that U is an embedding by composition. Namely,

- Set_{γ^*} *is an embedding.* Consider the map

  $$F : (A : \mathrm{Set}) \to (G \to_{\mathrm{Grp}} \mathrm{Aut}\, A) \to (X^* \to_{\mathrm{Grp}} \mathrm{Aut}\, A)$$

  obtained by precomposition by γ^*. Since γ^* is surjective we have that $F\,A$ is an embedding for every set A [27, Lemma 10.1.4]. By Lemma 14, we deduce that $\Sigma\,\mathrm{Set}\,.F$, which is Set_{γ^*}, is an embedding.
- Set_ι *is an embedding.* By universal property of X^*, given a set A, the map

  $$(X^* \to_{\mathrm{Grp}} \mathrm{Aut}\, A) \to (X \to \mathrm{Aut}\, A)$$

  obtained by precomposition by ι is an equivalence, and thus an embedding. Moreover, the property of being an isomorphism is a proposition, hence the forgetful map

  $$(X \to \mathrm{Aut}\, A) \to (X \to \mathrm{End}\, A)$$

  is an embedding because its fibers are propositions. Since embeddings are stable under composition, we deduce that the induced map

  $$F : (A : \mathrm{Set}) \to (X^* \to_{\mathrm{Grp}} \mathrm{Aut}\, A) \to (X \to \mathrm{End}\, A)$$

  is such that $F\,A$ is an embedding for every set A. By Lemma 14, the map $\Sigma\,\mathrm{Set}\,.F$, which is Set_ι, is thus an embedding. ◀

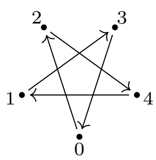
6 Cayley graphs

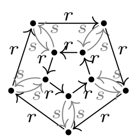
We have seen in Section 3 that a delooping of G can be obtained by further homotopy quotienting a delooping of X^*. The kernel of the map $\gamma^* : X^* \to G$ measures the defect of X^* from being G, which corresponds to the relations of the group. We show here that, under the delooping operation, those relations are precisely encoded by the Cayley graph [9, 19], a classical and useful construction in group theory which can be associated to any generated group.

The *Cayley graph* of G, with respect to the generating set X, is the directed graph whose vertices are the elements of G, and such that for every vertex $g : G$ and generator $x : X$, we have an edge $g \to gx$. In homotopy type theory, it is thus natural to represent it as the higher inductive type $\mathrm{C}(X, G)$ defined as

$$\begin{aligned} &\mathsf{vertex} : G \to \mathrm{C}(X,G) \\ &\mathsf{edge} \;\; : (g : G)(x : X) \to \mathsf{vertex}\, g = \mathsf{vertex}(gx) \end{aligned}$$

For instance, the Cayley graphs associated to $\mathbb{Z}_5$ (with 2 as generator) and D_5 (with r and s as generators) are respectively





Our main result in this section is that this type satisfies the following property. We recall that the delooping operation is functorial, see Appendix B and [29]: in particular, we can deloop morphisms. We also recall that the *kernel* of a map $f : A \to B$ with B pointed is its fiber at the distinguished element $\star$ of B.

▶ **Theorem 16** (`Cayley.Cayley-ker`). *The type* $\mathrm{C}(X, G)$ *is the kernel of the function* $\mathrm{B}\,\gamma^* : \mathrm{B}\,X^* \to \mathrm{B}\,G$ *induced by* γ*, i.e. we have*

$$\mathrm{C}(X,G) \quad = \quad \Sigma(x : \mathrm{B}\,X^*).(\star = \mathrm{B}\,\gamma^*(x)).$$

Proof. We define the type family $P : \mathrm{B}\,X^* \to \mathcal{U}$ by $P(x) = (\star = \mathrm{B}\,\gamma^*(x))$. Remember that $\mathrm{B}\,X^*$ admits a description as a coequalizer, see (1) and Proposition 1. Hence, by the flattening lemma for coequalizers (see Lemma 25 and [27, Section 6.12]), we have a coequalizer of total spaces

$$\Sigma X.P(\star) \underset{(x,p)\mapsto(\star,P_p^{\rightarrow}(x))}{\overset{(x,p)\mapsto(\star,p)}{\rightrightarrows}} \Sigma 1.P(\star) \dashrightarrow \Sigma\,\mathrm{B}\,X^*.P$$

By using the properties of transport in path spaces [27, Theorem 2.11.3], it can be shown that the bottom map sends (x, p) to $(\star, p \cdot \mathrm{B}\,\gamma^*(x))$. Moreover, $\mathrm{B}\,\gamma^*$ is pointed, so $P(\star)$ is equal to $\Omega\,\mathrm{B}\,G$, i.e. G, and we have the following coequalizer:

$$X \times G \underset{(x,g)\mapsto gx}{\overset{(x,g)\mapsto g}{\rightrightarrows}} G \dashrightarrow \Sigma\,\mathrm{B}\,X^*.P$$

If follows that $\Sigma\,\mathrm{B}\,X^*.P$ consists in $|G|$ points and a path $g = gx$ for each pair $(x, g) : X \times G$, and is therefore equal to the Cayley graph $\mathrm{C}(X, G)$. ◀

The above result can be interpreted as stating that we have a fiber sequence (see [27, Section 8.4])

$$\mathrm{C}(X, G) \longrightarrow \mathrm{B}\,X^* \longrightarrow \mathrm{B}\,G$$

which encodes the fact that we have an action of G on its Cayley graph, whose homotopy quotient $\mathrm{C}(X, G)/\!\!/G$ is $\mathrm{B}\,X^*$, see [21, Proposition 16]. Hence, we recover the canonical action of G on its Cayley graph.

Relations. The long exact sequence of homotopy groups induced by the above fiber sequence [27, Theorem 8.4.6] implies in particular that we have the following short exact sequence of groups

$$0 \longrightarrow \Omega\,\mathrm{C}(X, G) \longrightarrow X^* \longrightarrow G \longrightarrow 0$$

which shows that $\Omega\,\mathrm{C}(X, G)$ is the (free) group encoding relations of G with respect to X. Indeed, we have that $\mathrm{C}(X, G) = \mathrm{B}\,R^*$ where R is a choice of $|G| \times (|X| - 1) + 1$ relations: those are the loops in the Cayley graph after contracting $|G| - 1$ edges to obtain a wedge of circles. In some sense, Theorem 16 provides an internalization of the fact that G is presented by $\langle X \mid R\rangle$, contrasting with the point of view developed in Section 3.

The Cayley complex and higher variants. We now briefly explain that we can extend the previous construction in higher dimensions in order to define internally a type corresponding to the classical *Cayley complex* [19]. Suppose given a presentation $P := \langle X \mid R\rangle$ for G and write $\mathrm{B}_2\,P$ for the 2-skeleton of the type $\mathrm{B}\,P$ defined in Section 3 (i.e. the type generated by $\star$, gen, and rel, but without the truncation trunc). As in previous section, this type can be considered as an approximation of $\mathrm{B}\,P$ (lacking the truncation) and we would like to measure the difference between the two types. The *Cayley complex* $\mathrm{C}\,P$ associated to the presentation P is the inductive type defined by

$$\begin{aligned}
&\mathsf{vertex} : G \to \mathrm{C}\,P\\
&\mathsf{edge} \quad : (g : G)(x : X) \to \mathsf{vertex}\,g = \mathsf{vertex}(gx)\\
&\mathsf{cell} \quad : (g : G)(r : R) \to (\mathsf{edge}\,g)^*(\pi(r)) = (\mathsf{edge}\,g)^*(\pi'(r))
\end{aligned}$$

where, given $g : G$ and $u := x_1x_2 \dots x_n : X^*$, we have that $(\mathsf{edge}\,g)^*\,u$ is the path

$$\mathsf{v}\,g \overset{\mathsf{e}\,g\,x_1}{=\!=} \mathsf{v}(gx_1) \overset{\mathsf{e}(gx_1)\,x_2}{=\!=} \mathsf{v}(gx_1x_2) \overset{\cdots}{=\!=} \cdots \overset{\cdots}{=\!=} \mathsf{v}(gx_1 \dots x_{n-1}) \overset{\mathsf{e}\,g(x_1\dots x_{n-1})\,x_n}{=\!=} \mathsf{v}(gx_1 \dots x_n)$$

where v (resp. e) is a short notation for vertex (resp. edge). We will detail in future works the proof of the following result, which can be performed using the flattening lemma for pushouts, as in Theorem 16:

▶ **Theorem 17.** *We have a fiber sequence* $\mathrm{C}\,P \longrightarrow \mathrm{B}_2\,P \longrightarrow \mathrm{B}\,G$.

In fact, this resolution-like process can be iterated in order to obtain better and better approximations $\mathrm{B}_n\,P$ of $\mathrm{B}\,G$, and higher Cayley complexes as the fibers of the canonical maps $\mathrm{B}_n\,P \to \mathrm{B}\,G$. Moreover, the join construction [23, 24] provides a way to automate this task, see for instance [7, 21].

7 Future works

We have presented two ways to improve in practice the known constructions of deloopings of groups when we have a presentation of the group. This work is part of a larger investigation of "efficient" models of groups deloopings, in the sense that we can compute effectively with those. In particular, the construction of the infinite real projective space performed by Buchholtz and Rijke [7], provides a cellular description of $\mathrm{B}\,\mathbb{Z}_2$ (which is better than the usual ones obtained by generic methods because it consists in a *non-recursive* higher inductive type). In other current work, we refine their approach in order to construct lens spaces and thus obtain a cellular version of $\mathrm{B}\,\mathbb{Z}_n$ for every natural number n, as well as efficient representations of deloopings of other classical groups [21]. More generally, the formalization of group theory in univalent foundations is still under heavy investigation [2], and we aim at developing general techniques to construct efficient representations of (internal) groups in homotopy type theory, which would open the way to cohomological computations [8, 6, 4] or the definition of group actions on higher types (as a generalization of group actions on sets). On another note, higher-inductive types play a role in homotopy type theory analogous to the one of polygraphs for strict higher categories: numerous techniques have been developed for those [1], notably based on rewriting, and we plan to adapt them in this setting, following first developments of [16].

References

1 Dimitri Ara, Albert Burroni, Yves Guiraud, Philippe Malbos, François Métayer, and Samuel Mimram. Polygraphs: From rewriting to higher categories. To appear, 2023. arXiv:2312.00429.

2 Marc Bezem, Ulrik Buchholtz, Pierre Cagne, Bjørn Ian Dundas, and Daniel R. Grayson. Symmetry. URL: https://github.com/UniMath/SymmetryBook.

3 Marc Bezem, Ulrik Buchholtz, Daniel R Grayson, and Michael Shulman. Construction of the circle in UniMath. *Journal of Pure and Applied Algebra*, 225(10):106687, 2021. doi:10.1016/j.jpaa.2021.106687.

4 Guillaume Brunerie, Axel Ljungström, and Anders Mörtberg. Synthetic integral cohomology in cubical Agda. In *30th EACSL Annual Conference on Computer Science Logic (CSL 2022)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.CSL.2022.11.

5 Ulrik Buchholtz, J Daniel Christensen, Jarl G Taxerås Flaten, and Egbert Rijke. Central H-spaces and banded types. Preprint, 2023. arXiv:2301.02636.

6 Ulrik Buchholtz and Kuen-Bang Hou. Cellular cohomology in homotopy type theory. *Logical Methods in Computer Science*, 16, 2020. doi:10.23638/LMCS-16(2:7)2020.

7 Ulrik Buchholtz and Egbert Rijke. The real projective spaces in homotopy type theory. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–8. IEEE, 2017. doi:10.5555/3329995.3330081.

8 Evan Cavallo. *Synthetic cohomology in homotopy type theory*. PhD thesis, MA thesis, 2015.

9 Arthur Cayley. Desiderata and suggestions. No. 2 – The theory of groups: graphical representation. *American journal of mathematics*, 1(2):174–176, 1878.

10 Camil Champin and Samuel Mimram. Delooping generated groups in homotopy type theory. Software, swhId: `swh:1:dir:4fc863802e40f99733703893622a7aa23c50c308` (visited on 2024-06-20). URL: `https://github.com/smimram/generated-deloopings-agda`.

11 Thierry Coquand and Nils Anders Danielsson. Isomorphism is equality. *Indagationes Mathematicae*, 24(4):1105–1120, 2013. `doi:10.1016/j.indag.2013.09.002`.

12 Michel Demazure and Pierre Gabriel. *Groupes Algébriques, Tome 1*. North-Holland Publishing Company, 1970.

13 John RJ Groves. Rewriting systems and homology of groups. In *Groups—Canberra 1989: Australian National University Group Theory Program 1989*, pages 114–141. Springer, 2006. `doi:10.1007/BFb0100735`.

14 Krzysztof Kapulkin and Peter LeFanu Lumsdaine. The simplicial model of Univalent Foundations (after Voevodsky). *Journal of the European Mathematical Society*, 23(6):2071–2126, 2021. `doi:10.4171/jems/1050`.

15 Nicolai Kraus and Thorsten Altenkirch. Free higher groups in homotopy type theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 599–608, 2018. `doi:10.1145/3209108.3209183`.

16 Nicolai Kraus and Jakob von Raumer. A rewriting coherence theorem with applications in homotopy type theory. *Mathematical Structures in Computer Science*, 32(7):982–1014, 2022. `doi:10.1017/S0960129523000026`.

17 Daniel R Licata and Eric Finster. Eilenberg-MacLane spaces in homotopy type theory. In *Proceedings of the 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)*, pages 1–9, 2014. `doi:10.1145/2603088.2603153`.

18 Peter LeFanu Lumsdaine and Michael Shulman. Semantics of higher inductive types. *Mathematical Proceedings of the Cambridge Philosophical Society*, 169(1):159–208, 2020. `doi:10.1017/S030500411900015X`.

19 Roger C Lyndon and Paul E Schupp. *Combinatorial group theory*, volume 188. Springer, 1977.

20 Per Martin-Löf. *Intuitionistic type theory*, volume 1 of *Studies in proof theory*. Bibliopolis, 1984.

21 Émile Oleon and Samuel Mimram. Delooping cyclic groups with lens spaces in homotopy type theory. Accepted at LICS 2024 conference, 2024.

22 Henri Poincaré. *Analysis situs*. Gauthier-Villars Paris, France, 1895.

23 Egbert Rijke. The join construction. Preprint, 2017. `arXiv:1701.07538`.

24 Egbert Rijke. *Classifying Types*. PhD thesis, Carnegie Mellon University, July 2018. `arXiv:1906.09435`.

25 Egbert Rijke. Introduction to homotopy type theory. Preprint, 2022. `arXiv:2212.11082`.

26 The Agda Community. Cubical Agda Library, July 2023. URL: `https://github.com/agda/cubical`.

27 The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study, 2013. URL: `https://homotopytypetheory.org/book/`.

28 Andrea Vezzosi, Anders Mörtberg, and Andreas Abel. Cubical Agda: A dependently typed programming language with univalence and higher inductive types. *Journal of Functional Programming*, 31, 2021. `doi:10.1145/3341691`.

29 David Wärn. Eilenberg–maclane spaces and stabilisation in homotopy type theory. *Journal of Homotopy and Related Structures*, 18(2):357–368, 2023. `doi:10.1007/s40062-023-00330-5`.

30 David Wärn. Path spaces of pushouts. Preprint, 2023. `arXiv:2402.12339`.

A Omitted proofs

Proof of Proposition 8. The two functions

$$\phi : \operatorname{Aut} \mathrm{P}_G \to G \qquad f \mapsto f(1) \qquad\qquad \psi : G \to \operatorname{Aut} \mathrm{P}_G \qquad x \mapsto y \mapsto yx$$

are group morphisms. Namely, given $f, g : \operatorname{Aut} \mathrm{P}_G$, we have

$$\phi(g \circ f) = g \circ f(1) = g(f(1)1) = f(1)g(1) = \phi(f)\phi(g) \qquad \phi(\mathrm{id}) = \mathrm{id}(1) = 1$$

and given $x, y : G$, we have for every $z : G$,

$$\psi(xy)(z) = z(xy) = (zx)y = \psi(y) \circ \psi(x)(z) \qquad \psi(1)(x) = x1 = \mathrm{id}(x)$$

Moreover, they are mutually inverse. Namely, given $f : \operatorname{Aut} \mathrm{P}_G$ and $x : G$, we have

$$\psi \circ \phi(f)(x) = xf(1) = f(x1) = f(x) \qquad \phi \circ \psi(x) = 1x = x$$

We thus have $\operatorname{Aut} \mathrm{P}_G \simeq G$ and we conclude by univalence. ◀

Proof of Lemma 14. By definition, the map $\Sigma A.f$ is an embedding iff for every (a, x) and (a', x') in $\Sigma A X$, the induced map

$$(a, x) = (a', x') \to (a, f\, a\, x) = (a', f\, a'\, x')$$

is an equivalence. By the characterization of equalities in Σ-types [27, Theorem 2.7.2], this map corresponds to a map

$$(\Sigma(p : a = a').P_p^{\to}(x) = x') \to (\Sigma(p : a = a').Q_p^{\to}(f\, a\, x) = f\, a'\, x')$$

By [27, Theorem 4.7.7], this is an equivalence if and only if the fiber map

$$(P_p^{\to}(x) = x') \to (Q_p^{\to}(f\, a\, x) = f\, a'\, x')$$

is an equivalence for every $p : a = a'$. By path induction, this is true if and only if

$$(f\, a)^{=} : x = x' \to f\, a\, x = f\, a\, x'$$

is an equivalence for all $a : A$, and $x, x' : Xa$. By definition, this is the requirement that $f\, a$ is an embedding for all $a : A$. ◀

B Equivalence between internal and external groups

Functoriality of delooping. One of the main properties of the delooping operation is that it is a "local inverse" to taking loop spaces in the following sense:

▶ **Proposition 18.** *Given pointed connected groupoids A and B and a group morphism $f : \Omega\, A \to \Omega\, B$, there is a unique pointed morphism $g : A \to B$ such that $\Omega\, g = f$.*

Proof. By [29, Corollary 12], with $n = 0$, we have that the type $\Sigma(g : A \to B).\, \Omega\, g = f$ is equivalent to $(p, q : \star_A = \star_A) \to f(p \cdot q) = f(p) \cdot f(q)$. This type is a proposition because $\Omega\, B$ is a set (because B is a groupoid), and inhabited (because f is a morphism of groups), and thus contractible. ◀

As an immediate consequence of the above lemma, deloopings are unique:

▶ **Proposition 19.** *Given two deloopings $\mathrm{B}\, G$ and $\mathrm{B}'\, G$ of a group G, we have $\mathrm{B}\, G = \mathrm{B}'\, G$.*

Given a group morphism $f : G \to H$ such that both G and H admit deloopings (and this actually always holds by Theorem 9), the *delooping* of f is the morphism

$$\mathrm{B}\, f : \mathrm{B}\, G \to \mathrm{B}\, H$$

associated, by Proposition 18, to the morphism $\mathrm{d}_H^{\leftarrow} \circ f \circ \mathrm{d}_G^{\rightarrow} : \Omega\, \mathrm{B}\, G \to \Omega\, \mathrm{B}\, H$. By Proposition 18, this operation is functorial in the sense that that it preserves identities and composition.

Equivalence between the two points of view. Although this is not central in this article, we shall mention here the fundamental equivalence provided by the above constructions; details can be found in [2]. We write IntGroup for the type of internal groups, i.e. pointed connected groupoids.

▶ **Theorem 20.** *The maps* Ω : IntGroup $\to$ Group *and* B : Group $\to$ IntGroup *form an equivalence of types.*

Proof. Given a group G, we have $\Omega\, \mathrm{B}\, G = G$ by definition of $\mathrm{B}\, G$. Given an internal group A, we have $\mathrm{B}\, \Omega\, A \simeq A$ by Proposition 19. ◀

The above theorem thus states looping and delooping operators allow us to go back and forth between the external and the internal point of view of group theory in homotopy type theory. Note that the torsor construction only gives a delooping in a larger universe than the original group unless one makes additional assumptions such as the *replacement axiom* [25, Axiom 18.1.8].

Internal group actions. In a similar way that the traditional notion of group admits an internal reformulation (Section 2), the notion of action also admits an internal counterpart which can be defined as follows. Given a group G, an *internal* action of G on a set A is a function

$$\alpha : \mathrm{B}\, G \to \mathrm{Set}$$

such that $\alpha(\star) = A$. Since Set is a groupoid [27, Theorem 7.1.11], by Theorem 20, we have equivalences of types

$$(\mathrm{B}\, G \to \mathrm{Set}) \quad \simeq \quad (\Omega\, \mathrm{B}\, G \to \Omega(\mathrm{Set}, A)) \quad \simeq \quad (G \to \mathrm{Aut}\, A)$$

which show that internal group actions correspond to external ones: the delooping operator internalizes an external group action, and the looping operator externalizes an internal group action.

C Connected components

We define the *connected component* of the pointed type A as the type of points which are merely connected to the distinguished point of A. This type is noted $\mathrm{Comp}\, A$ (or $\mathrm{Comp}(A, \star)$ when we want to specify the distinguished element $\star$). Formally,

$$\mathrm{Comp}\, A \quad := \quad \Sigma(x : A).\|\star = x\|_{-1}$$

This type is canonically pointed by $(\star, |\,\mathrm{refl}\,|_{-1})$. This construction deserves its name because it produces a connected space, whose geometry is the same as the original space around the distinguished point, as shown in the following two lemmas.

▶ **Lemma 21** (`Comp.isConnectedComp`). *The type* $\mathrm{Comp}\, A$ *is connected.*

Proof. It can be shown that a type X is connected precisely when both $\|X\|_{-1}$ and $(x, y : X) \to \|x = y\|_{-1}$ are inhabited, i.e. when X merely has a point and any two points are merely equal [27, Exercise 7.6]. In our case, the type $\operatorname{Comp} A$ is pointed and thus $\|\operatorname{Comp} A\|_{-1}$ holds. Moreover, suppose that there are two points (x, p) and (y, q) in $\operatorname{Comp} A$ with $x, y : A$, $p : \|\star = x\|_{-1}$ and $q : \|\star = y\|_{-1}$. Our goal is to show that $\|(x, p) = (y, q)\|_{-1}$ holds, which is a proposition, so by elimination of propositional truncation, we can therefore assume that p (resp. q) has type $\star = x$ (resp. $\star = y$). Hence, we can construct a path $p^- \cdot q$ of type $x = y$, and therefore $(x, p) = (y, q)$ because the second components belong to a proposition by propositional truncation. We conclude that $\|(x, p) = (y, q)\|_{-1}$ and finally that $\operatorname{Comp} A$ is connected. ◀

▶ **Lemma 22** (`Comp.loopCompIsLoop`). *We have* $\Omega \operatorname{Comp} A = \Omega A$.

Proof. We begin by showing that the type

$$\Sigma((x, t) : \operatorname{Comp} A).(\star = x) \tag{7}$$

is contractible. In order to do so, observe that we have the following equivalence of types:

$$\begin{aligned}\Sigma((x, t) : \operatorname{Comp} A).(\star = x) &\simeq \Sigma((x, t) : \Sigma(x : A).\|\star = x\|_{-1}).(\star = x)\\ &\simeq \Sigma(x : A).(\|\star = x\|_{-1}) \times (\star = x)\\ &\simeq \Sigma((x, p) : \Sigma(x : A).(\star = x)).\|\star = x\|_{-1}\end{aligned}$$

using classical associativity and commutativity properties of Σ-types. Moreover, the type $\Sigma(x : A).(\star = x)$ is contractible [27, Lemma 3.11.8], therefore the whole type on the last line is a proposition (as a sum of propositions over a proposition), and therefore also the original type (7). We write $\star'$ for the element $(\star, \|\operatorname{refl}_\star\|_{-1})$ of $\operatorname{Comp} A$. The type (7) is pointed by the canonical element $(\star', \operatorname{refl})$ and thus contractible as a pointed proposition.

We have a morphism

$$\begin{array}{rccc} F : ((x, t) : \operatorname{Comp} A) & \to & (\star' = (x, t)) & \to & (\star = x)\\ (x, t) & & p & \mapsto & \pi^=(p)\end{array}$$

sending a path p to the path obtained by applying the first projection. It canonically induces a morphism

$$\begin{aligned}\Sigma((x, t) : \operatorname{Comp} A).(\star' = (x, t)) &\to \Sigma((x, t) : \operatorname{Comp} A).(\star = x)\\ ((x, t), p) &\mapsto ((x, t), \pi^=(p))\end{aligned}$$

between the corresponding total spaces. Since the left member is contractible (by [27, Lemma 3.11.8] again) and the right member is contractible (as shown above), this is an equivalence. By [27, Theorem 4.7.7], for every $x : \operatorname{Comp} A$, the fiber morphism Fx is also an equivalence. In particular, with x being $\star'$, we obtain $\Omega \operatorname{Comp} A \simeq \Omega A$ (as a type) and we can conclude by univalence. Note that the equivalence preserves the group structure so that the equality also holds in groups. ◀

As a direct corollary of the two above lemmas, we have:

▶ **Proposition 23.** *Given a pointed groupoid* A, $\operatorname{Comp} A$ *is a delooping of* ΩA.

▶ Remark 24. Some people write $\operatorname{Aut} A$ for ΩA and the above proposition states that we have $\operatorname{B} \operatorname{Aut} A = \operatorname{Comp} A$. For this reason, the (confusing) notation $\operatorname{BAut} A$ is also found in the literature for $\operatorname{Comp} A$.

D The flattening lemma

We recall here the classical flattening lemma, see [27, Section 6.12] for a more detailed presentation and proof.

▶ **Lemma 25** (Flattening for coequalizers). *Suppose given a coequalizer*

$$A \underset{g}{\overset{f}{\rightrightarrows}} B \overset{h}{\dashrightarrow} C$$

with $p : h \circ f = h \circ g$*, and a type family* $P : C \to \mathcal{U}$*. Then the diagram*

$$\Sigma A.(P \circ h \circ f) \underset{\Sigma g.e}{\overset{\Sigma f.(\lambda\_.\,\mathrm{id}\,)}{\rightrightarrows}} \Sigma B.(P \circ h) \overset{\Sigma h.(\lambda\_.\,\mathrm{id}\,)}{\dashrightarrow} \Sigma C.P$$

is a coequalizer, where the map

$$e : (a : A) \to P(h(f(a))) \to P(h(g(a)))$$

is induced by transport along p *by*

$$e\, a\, x := P^{\to}_{p(a)}(x).$$

Note that there is a slight asymmetry: we could have formulated a similar statement with $\Sigma A.(P \circ h \circ g)$ as left object.

Machine-Checked Categorical Diagrammatic Reasoning

Benoît Guillemet
École normale supérieure Paris-Saclay, France

Assia Mahboubi
Nantes Université, École Centrale Nantes, CNRS, INRIA, LS2N, UMR 6004, France
Vrije Universiteit Amsterdam, The Netherlands

Matthieu Piquerez
Nantes Université, École Centrale Nantes, CNRS, INRIA, LS2N, UMR 6004, France

Abstract

This paper describes a formal proof library, developed using the Coq proof assistant, designed to assist users in writing correct diagrammatic proofs, for 1-categories. This library proposes a deep-embedded, domain-specific formal language, which features dedicated proof commands to automate the synthesis, and the verification, of the technical parts often eluded in the literature.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Interactive theorem proving, categories, diagrams, formal proof automation

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.7

Related Version *Previous Version*: `https://arxiv.org/abs/2402.14485`
Previous Version: `https://inria.hal.science/hal-04471683v4/`

Supplementary Material
Software (Source Code): `https://gitlab.inria.fr/mpiquere/coq-diagram-chasing` [10]
archived at `swh:1:dir:a730f6752583959dd173217435f10e0a491ff957`

Funding This work has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 101001995).

Acknowledgements The authors would like to thank the anonymous reviewers for their valuable comments and suggestions.

1 Introduction

Abstract nonsense, a non derogatory expression attributed to Steenrod, usually refers to the incursion of categorical methods for a proof step deemed both technical and little informative, and therefore often succinctly described. *Diagrams* are typically drawn in this case, so as to guide the intuition of the audience, and help visualize the existence of certain morphisms or objects, identities between composition of morphisms, etc.

Formally, a categorical diagram is a functor $F\colon J \to \mathcal{C}$, with J a small category called the *shape* of the diagram [20]. Diagrams are depicted as directed multi-graphs, also called *quivers*, whose vertices are decorated with the objects of $\mathcal{C}$ and whose arrows each represent a certain morphism, between the objects respectively decorating its source and its target. A directed path in the diagram is hence associated with a chain of composable arrows and a diagram *commutes* when all directed paths with same source and target lead to equal compositions. Equalities between compositions of morphisms thus correspond to the commutativity of certain sub-diagrams of a larger diagram. Chasing commutative sub-diagrams in a larger diagram provides an elegant alternative to equational reasoning, when the latter becomes overly technical. Diagrams actually play a central role in category theory, for they provide

such an efficient way of delivering convincing enough proofs. Some classical textbooks introduce diagrams as early as in their introduction chapter [17], while others devote an entire section to diagrammatic categorical reasoning [23, Section 1.6] [14, Session 17]. The following diagrammatic proof of Lemma 1 provides a toy illustrative example of this technique.

▶ **Lemma 1.** *Let $\mathcal{C}$ be a category. For any morphism f and g such that $g \circ f \in \mathrm{Hom}(\mathcal{C})$, if $g \circ f$ is a monomorphism, then so is f.*

Proof. Consider $f\colon A \to B$, $g\colon B \to C$ and $g \circ f\colon A \rightarrowtail C$ morphisms in a category $\mathcal{C}$. Here is a very detailed diagrammatic proof, taking place in the diagram of Figure 1.

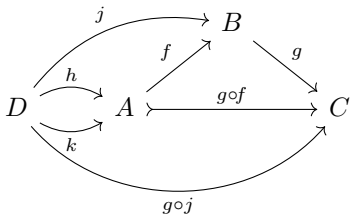


Figure 1 Diagrammatic proof that if $g \circ f$ is a monomorphism, then so is f.

We need to prove that for any two other morphisms $h, k\colon D \rightrightarrows A$ in $\mathrm{Hom}(\mathcal{C})$ such that $f \circ h = f \circ k$, we have $h = k$, i.e., that the two-arrow diagram h, k commutes. By hypothesis on h and k, there is an arrow j such that the triangle diagrams respectively formed by arrows h, f, j and k, f, j both commute. By definition of composition, the triangle diagrams formed by arrows $f, g, g \circ f$ and $j, g, g \circ j$ also commute. As a consequence, both triangle diagrams formed by $h, g \circ f, g \circ j$ and $k, g \circ f, g \circ j$ commute. The conclusion follows because $g \circ f$ is a monomorphism. ◀

Diagram chasing actually refers to a central technique to homological algebra [16, 20], operating on diagrams over Abelian categories and used for proving the existence, injectivity, surjectivity of certain morphisms, the exactness of some sequences, etc. Classic examples of proofs by diagram chasing include that of the *five lemma* or of the *snake lemma* [16]. However, complex diagram chases (see, e.g., [22, p.338]) only remain readable at the price of hiding non-trivial technical arguments and are, in practice, challenging to rigorously verify by hand. Typically, the reader of a diagrammatic proof is asked to solve instances of variable difficulty of a decision problem hereafter referred to as the *commerge* problem: *Given a collection of sub-diagrams of a larger diagram which commute, must the entire diagram commute?* In addition, proofs may resort to non-trivial duality arguments, in which the reader has to believe a certain property about diagrams in any Abelian category remains true after reversing all the involved arrows.

The long-term objective of the present work is thus to build a computer-aided instrument for devising both fluent and reliable categorical diagrammatic reasoning, for 1-categories. The present article describes the implementation of the core of such a tool, as a library for the Coq proof assistant [24]. The design of this library follows two main design principles. First, it aims at being independent from any specific library of formalized category theory or abstract algebra, but rather usable as a helper for any existing one. This independence is achieved by formulating the propositions about diagrams as formulas of a deep-embedded language. Specific libraries then lead to specific interpretation functions, turning deep-embedded formulas into actual statements. Second, it strives to feature enough automation tools for synthesizing the bureaucratic parts of proofs "by abstract nonsense", and formal proofs thereof. The main contributions presented here are thus:

- a formalization of a deep-embedded first-order language for category theory, geared towards diagrammatic proofs, together with a generic formalized definition of categorical diagrams. This formalized material is based on a variant of the paper definitions introduced in a previous work by two of the authors [18];
- automation support for proofs by duality in the corresponding reified proof system, that we compare with another method to deal with duality that was introduced in [18];
- automation support for the *commerge* problem.

The corresponding code is available at the following url [10].

The rest of the article is organized as follows. Section 2 fixes some vocabulary and describes the corresponding formal definitions. Section 3 describes the deep-embedded formalization of the first-order language, and of the related reified proof system. Section 4 explains the algorithms involved in automating commutativity proofs. The formalized version of the proof of Lemma 1 is explained in detail in Section 5. We conclude in Section 6 by discussing related work and a few perspectives.

2 Preliminaries

This section fixes some definitions and notations, and introduces their formalized counterpart when relevant. Some of them coincide with the preliminaries of our previous text [18], for the purpose of being self-contained. Their formalized counterpart is novel. Throughout this article, we use the word *category* for 1-categories. By default, we do not display implicit arguments in Coq terms.

In all what follows, $\mathbb{N} \coloneqq \{0, 1, \dots\}$ refers to the set of non-negative integers, represented in Coq by the type `nat`, from its standard library. We also use the standard polymorphic type `list` for finite sequences, equipped with the library on sequences distributed by the Mathematical Components [19] library. Some names thus slightly differ from those present in the standard library. For instance, the size $|l|$ of a finite sequence l, formalized as `l : list T`, is `size l`, instead of the standard `length l`. We document in comments the definitions we use from this library when their names are not self-explanatory. We recall that `b1 && b2` is the standard notation for the (boolean) conjunction of two boolean values `b1 b2 : bool`. If $n \in \mathbb{N}$, then $[n]$ denotes the finite collection $\{0, \dots, n-1\}$, which is implemented by the sequence `iota 0 n : list nat`.

▶ **Definition 2** (General quiver, dual). *A* general quiver $\mathcal{Q}$ *is a quadruple* $(V_\mathcal{Q}, A_\mathcal{Q}, s_\mathcal{Q} \colon A_\mathcal{Q} \to V_\mathcal{Q}, t_\mathcal{Q} \colon A_\mathcal{Q} \to V_\mathcal{Q})$ *where* $V_\mathcal{Q}$ *and* $A_\mathcal{Q}$ *are two sets. The element of* $V_\mathcal{Q}$ *are called the* vertices *of* $\mathcal{Q}$ *and the element of* $A_\mathcal{Q}$ *are called* arrows. *If* $a \in A_\mathcal{Q}$, $s_\mathcal{Q}(a)$ *is called the* source *of* a *and* $t_\mathcal{Q}(a)$ *is called its* target. *The* dual *of a quiver* $\mathcal{Q}$ *is the quiver* $\mathcal{Q}^\dagger \coloneqq (V_\mathcal{Q}, A_\mathcal{Q}, t_\mathcal{Q}, s_\mathcal{Q})$, *which swaps the source and the target maps of* $\mathcal{Q}$.

From now on, we casually call *quivers* the special case of quivers with $V_\mathcal{Q}$ a finite subset of $\mathbb{N}$ and with a finite number of arrows. The formal definition moreover assumes that vertices are labelled in order:

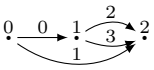
```
(* Data of a quiver *)
Record quiver : Type := quiver_Build {
  quiver_nb_vertex : nat;  (* number of vertices *)
  quiver_arc : list (nat * nat); (* sequence of arrows *) }.

(* Well-formedness condition for quivers :
  all arrows involved in A have a source and target in bound *)
Definition quiver_wf '(quiver_Build n A) : bool :=
  all (fun a => (a.1 < n) && (a.2 < n)) A.
```

```
(* The dual quiver of a quiver, with same vertices and reversed arrows *)
Definition quiver_dual '(quiver_Build n A) : quiver :=
  quiver_Build n (map (fun a => (a.2,a.1)) A).
```

An arrow of a quiver `q : quiver` is thus given by an element in the sequence `quiver_arc q`, itself a pair of integers giving its source and target respectively. Note that the index in the sequence matters, as the sequence may have duplicate. A formalized quiver is well-formed when the sources and targets of its arrows are in bound.

For the sake of readability, we use drawings to describe some quivers, as for instance:

For a quiver $\mathcal{Q}$ denoted by such a drawing, the convention is that $V_{\mathcal{Q}} = [\mathrm{card}(V_{\mathcal{Q}})]$ and $A_{\mathcal{Q}} = [\mathrm{card}(A_{\mathcal{Q}})]$, where $\mathrm{card}(A)$ denotes the cardinal of a finite set A. From left to right, the drawn vertices correspond to $0, 1, \ldots, \mathrm{card}(V_{\mathcal{Q}}) - 1$. Arrows are then numbered by sorting pairs $(s_{\mathcal{Q}}, t_{\mathcal{Q}})$ in increasing lexicographical order, as in:



▶ **Definition 3** (Morphism, embedding). *A* morphism of quivers $m \colon \mathcal{Q} \to \mathcal{Q}'$, *is the data of two maps* $m_V \colon V_{\mathcal{Q}} \to V_{\mathcal{Q}'}$ *and* $m_A \colon A_{\mathcal{Q}} \to A_{\mathcal{Q}'}$ *such that* $m_V \circ s_{\mathcal{Q}} = s_{\mathcal{Q}'} \circ m_A$ *and* $m_V \circ t_{\mathcal{Q}} = t_{\mathcal{Q}'} \circ m_A$. *Such a morphism is called an* embedding of quivers *if moreover both* m_V *and* m_A *are injective. In this case we write* $m \colon \mathcal{Q} \hookrightarrow \mathcal{Q}'$.

We also use drawings to denote embeddings. The black part represents the domain of the morphism, the union of black and gray parts represents its codomain. Here is an example of an embedding of the quiver into the quiver drawn above.

For the purpose of this work, we actually only need to define formally embedding morphisms, called sub-quivers, which select the relevant vertices and arrows from a quiver:

```
Record subquiver := subquiver_Build {
  subquiver_vertex : list nat; (* labels of the selected vertices *)
  subquiver_arc : list nat; (* indices of the selected arrows *) }.

(* Performs the expected selection of vertices and arrows *)
Definition quiver_restr '(subquiver_Build sV sA) : quiver -> quiver := (...)
```

Here as well, restrictions of quiver only make sense under well-formedness conditions:

```
(* Indices of the arrows to be selected are in bound *)
Definition quiver_restr_A_wf sA '(quiver_Build n A) : bool :=
    all (gtn (size A)) sA.

Definition quiver_restr_V_wf sV '(quiver_Build n A) : bool :=
  uniq sV && (* sV is duplicate-free *)
  all (gtn n) sV && (* all elements of sV are smaller than n *)
  all (fun a => (a.1 \in sV) && (a.2 \in sV)) A. (* any vertex involved in A is in sV*)

(* Well-formed condition on the restriction of a quiver *)
Definition quiver_restr_wf '(subquiver_Build sV sA) Q :=
  quiver_restr_A_wf sA Q && quiver_restr_V_wf sV (quiver_restr_A sA Q).
```

▶ **Definition 4** (Path-quiver). *The* path-quiver of length k, *denoted by* PQ_k, *is the quiver with* $k+1$ *vertices and* k *arrows* $([k+1], [k], \mathrm{id}, (i \mapsto i+1))$.

A path-quiver can be drawn as:

with at least one vertex. Such a path-quiver is called *nontrivial* if it has at least two vertices.

If $\mathcal{Q}$ is a general quiver, a morphism of the form $p\colon \mathrm{PQ}_k \to \mathcal{Q}$, for some k, is called a *path of* $\mathcal{Q}$ *from* u *to* v *of length* k, where $u \coloneqq p(0)$ and $v \coloneqq p(k)$. A general quiver is *acyclic* if any path of this quiver is an embedding.

If P is a nontrivial path-quiver, we define $\mathsf{st}_P\colon \bullet\!\bullet \hookrightarrow P$ to be the embedding mapping the first vertex on the leftmost vertex of P and the second vertex on the rightmost vertex of P. Two paths $p_1\colon P_1 \to \mathcal{Q}$, $p_2\colon P_2 \to \mathcal{Q}$ of $\mathcal{Q}$ *have the same extremities* if $p_1 \circ \mathsf{st}_{P_1} = p_2 \circ \mathsf{st}_{P_2}$. We denote by $\mathcal{BP}_{\mathcal{Q}}$ the set of pairs of paths of $\mathcal{Q}$ having the same extremities. Such a pair is called a *bipath*.

Paths in a formalized quivers are defined as a ternary relation between two vertices and a sequence of arrows:

```
(* Operations on lists:
  - (_ == _) is a generic boolean comparison test, in this case for lists of integers
  - rcons l x  is the list l followed by x
  - unzip[1 | 2] l is the list of fst (resp snd) elements of the list of pairs l
  - sub p A is the list of elements of A with index in p, in order *)
Definition path (A : list (nat * nat)) (u : nat) (p : list nat) (v: nat) : bool :=
  (* all elements in p are in bound *)
  all (gtn (size A)) p &&
  (* p selects in A a list of adjacent arrows from u to v *)
  (u :: unzip2 (sub p A) == rcons (unzip1 (sub p A)) v).
```

A path p from a vertex u to a vertex v can thus be concatenated to a path q from vertex v to a vertex w: when endpoints are obvious, we just write $p{\cdot}q$ the resulting path from u to w. We sometimes abuse notations and write $e{\cdot}p$ and $p{\cdot}e$ when one of the paths contains a single arrow e.

We now introduce a special case of binary relation on paths with same extremities in a quiver, called *path relations*. A path relation is an equivalence relation induced from a congruence on the corresponding free category to the quiver. Conversely, in a small category, the composition axiom induces a path relation on the underlying quiver. The formalized definition of path relations is actually independent from that of quiver. A path relation is just a family of equivalence relations on sequences of integers (the paths), indexed by pairs of integers (the endpoints), that are compatible with the concatenation of paths:

```
Record path_relation := {
  pi_r :> forall u v : nat, relation (list nat) ;
  pi_equiv : forall u v, equivalence _ (pi_r u v) ;
  pi_cat_stable : forall u v w p p' q q',
      pi_r u v p p' -> pi_r v w q q' -> pi_r u w (p ++ q) (p' ++ q') }.
```

Given a quiver and a path relation `r : path_relation`, `(pi_r r u v)` is expected to be a relation on the paths from vertex `u` to vertex `v`. Note that the path relation induced by a certain category on its underlying quiver only induces a partial collection of equivalence relations, indexed by the pairs of vertices in this quiver. The full relation on sequences, which relates any two sequences, can be used to complete this collection, so as to define a term of type `forall u v : nat, relation (list nat)`, and to avoid the need for otherwise cumbersome dependent types.

```
Inductive term :=
    | Var of nat (* variable, named with an integer, $k denotes term (Var k) *)
    | Restr of subquiver & term. (* the 'restr' symbol *)

Inductive formula :=
    | Forall of quiver & formula
    | Exists of quiver & formula
    | Imply of formula & formula (* Denoted with infix symbol -=> *)
    | And of formula & formula
    | FTrue (* Top atom *)
    | Commute of term (* the 'commute' predicate symbol *)
    | EqD of term & term.  (* the equality predicate symbol*)
```

Listing 1 Terms and formulas.

3 A two-level approach

3.1 Formulas and diagrams

Paraphrasing Mac Lane [17], many properties of category theory can be "unified and simplified by a presentation with diagrams of arrows". Categorical diagrammatic reasoning consists in transforming a proof of category theory into a proof about some quivers, decorated with the data of a certain category. Actually, once the appropriate quivers are drawn, the data themselves can be forgotten, but for the induced path relation, which is the only relevant information for a diagrammatic proof. In [18], we proposed a multi-sorted first-order language for category theory, geared towards diagrammatic reasoning: following the structure of a formula in this language constructs the quivers associated with the corresponding statement of category theory, encoded in the sorts of the variables. We recall the definition of this language:

▶ **Definition 5.** *We define a many-sorted signature* Σ *with sorts the collection of finite acyclic quivers. Signature* Σ *has one function symbol* $\mathrm{restr}_{m\colon \mathcal{Q}'\hookrightarrow\mathcal{Q}}$*, of arity* $\mathcal{Q}\to\mathcal{Q}'$*, per each* injective *quiver morphism* $m\colon \mathcal{Q}'\hookrightarrow\mathcal{Q}$ *between two quivers* $\mathcal{Q}$ *and* $\mathcal{Q}'$*, and one predicate symbol* $\mathrm{commute}_{\mathcal{Q}}$*, on sort* $\mathcal{Q}$*, for each finite acyclic quiver* $\mathcal{Q}$*.*

▶ **Example 6.** Writing the sorts of quantified variables as a subscript of the quantifier, here is for instance a predicate of arity $\bullet\!\to\!\bullet \times \bullet\!\to\!\bullet \times \bullet\!\to\!\bullet$ describing composite of arrows:

$$\mathrm{Comp}(x,y,z)\colon \quad \exists_{\bullet\to\bullet\to\bullet} w,\quad \mathrm{restr}_{\bullet\to\bullet\to\bullet}(w)\approx x \ \wedge\ \mathrm{restr}_{\bullet\to\bullet\to\bullet}(w)\approx y$$
$$\wedge\ \mathrm{restr}_{\bullet\to\bullet\to\bullet}(w)\approx z \ \wedge\ \mathrm{commute}(w)$$

▶ **Example 7.** Here is a predicate of arity $\bullet\!\to\!\bullet$ describing monomorphisms:

$$\mathrm{Mono}(x)\colon \quad \forall_{\bullet\rightrightarrows\bullet} w,\quad \mathrm{restr}_{\bullet\rightrightarrows\bullet}(w)\approx x \ \Rightarrow\ \mathrm{commute}(\mathrm{restr}_{\bullet\rightrightarrows\bullet}(w))$$
$$\Rightarrow \mathrm{commute}(\ \mathrm{restr}_{\bullet\rightrightarrows\bullet}(w)) \ \Rightarrow\ \mathrm{commute}(\mathrm{restr}_{\bullet\rightrightarrows\bullet}(w))$$

Listing 1 is the formalized counterpart of Definition 5. A term `t : term` is thus either a variable `Var n`, named with a natural number `n`, or of the form `Restr m t` for `t` a term and `m` a sub-quiver, seen as a morphism of quivers. Observe that the source quiver of this morphism is left undefined – it only becomes explicit when the term is evaluated. Type `formula` defines a first-order logic whose atoms stand for equality, commutativity or true. Quantifiers bind de Bruijn indexes and are annotated with a quiver, the sort of the bound

variable. Note that theory Σ enforces the use of acyclic quivers. Computing the sort of a term thus requires first annotating each of its variables with a sort. The sort of a term `Restr m t` is the quiver obtained by restricting the sort of term `t` using `m`.

Given a list `l` of quivers, providing a sort to each of its variable, a term `t : term` is well-formed in this context, written `term_wf l t`, if `l` is long enough and if every subterm of `t` has a well-formed sort. In a closed formula, the sort of a variable is read on the corresponding quantifier. More generally, a formula `f : formula` is well-formed in a context `l`, written `formula_wf l f`, if `l` is long enough to provide a sort to each free variable in `f` and if every term appearing in `f` has a well formed sort.

Here is the corresponding formalized predicate to Example 7.

```
Definition monoQ := quiver_Build 3 [:: (0,1);(0,1);(0,2);(1,2)]. (* This is ·⇉·→· *)
Definition mapQ := quiver_Build 2 [:: (0,1)]. (* This is ·→· *)

(* Lambda_arc constructs a predicate from a sequence of quivers and a formula, the first
argument is the arity, providing sorts for the free variables of the second, in order.*)
Definition monoF : predicate :=
  Lambda_arc [:: mapQ] (* the one-element arity sequence *)
  (Forall monoQ (
    EqD (Restr {sA [:: 3]} $0) $1
    -=> Commute (Restr {sA [:: 0 ; 2 ; 3]} $0)
    -=> Commute (Restr {sA [:: 1 ; 2 ; 3]} $0)
    -=> Commute (Restr {sA [:: 0 ; 1]} $0))) .
```

The interpretation of a term `f : formula` as a Coq statement, in sort `Prop`, is relative to a formal *diagram*, which is by definition an instance of the following structure `diagram_type`:

```
Record diagram_package (diagram : Type) := diagram_Pack {
  diagram_to_quiver : diagram -> quiver; (* underlying quiver of a diagram *)
  diagram_restr : subquiver -> diagram -> diagram; (* restriction *)
  eqD : equivalence diagram; (* setoid relation on type diagram *)
  eq_comp : diagram -> path_relation; (* path relation *) }.

Structure diagram_type := diagram_type_Build {
  diagram_sort :> Type; (* a diagram_type coerces to its carrier type *)
  diagram_to_package :> diagram_package diagram_sort; }.
```

A diagram is thus a term in the carrier type of an instance of structure `diagram_type`, which can be seen as a model of theory Σ. A diagram commutes when the associated path relation is *full*, i.e., any two paths in the underlying quiver with same source and target are related:

```
Definition commute (d : diagram_type) (D : d) :=
  path_total (diagram_to_quiver D) (eq_comp D).
```

Formalized diagrams can be defined from a formalization of categories, but not only. For instance, one can define an instance of `diagram_type` with the following carrier type:

```
Record zmod_diagram : Type := ZModDiagram {
  zmob : nat -> zmodType; (* labels of vertices *)
  zmmap : forall u v : nat, nat -> {additive zmob u -> zmob v}; (* labels of arrows *)
  zmdiag_to_quiver : quiver (* underlying quiver *)}.
```

where `zmodType` is a structure for Abelian groups in the Mathematical Components library, and `{additive A -> B}` is the type of morphisms between two Abelian groups `A` and `B`. Labelling functions `zmob` and `zmap`, respectively for vertices and for arrows, are total and thus require a default values for irrelevant arguments, albeit an arbitrary one. We can now explain how to turn a term `f : formula` into a Coq statement, given a sequence of diagrams:

```
Fixpoint formula_eval (d : diagram_type) (stack : list d) (f : formula) : Prop :=
  match f with
  | Forall Q f => forallD D :: diagram_on Q, formula_eval d (D :: stack) f
  | Exists Q f => existsD D :: diagram_on Q, formula_eval d (D :: stack) f
  | Imply f1 f2 => formula_eval d stack f1 -> formula_eval d stack f2
  | And f1 f2 => formula_eval d stack f1 /\ formula_eval d stack f2
  | FTrue => True
  | Commute t => if term_oeval stack t is Some D then commute D else False
  | EqD t1 t2 =>
    match term_oeval stack t1, term_oeval stack t2 with
    | Some DG1, Some DG2 => eqD DG1 DG2
    | _, _ => False
    end
  end.
```

where the notations `forallD D :: diagram_on Q, P` and `existsD D :: diagram_on Q, P` bind variable `D` in `P`, so as to quantify `P` over diagrams `D : d` with underlying quiver `diagram_to_quiver D` equal to `Q`. The evaluation `term_oeval stack t : option d` of a term `t` in context `stack` defaults to `None` when the context `stack` is too small and otherwise computes, when possible, the prescribed restriction of the diagrams. Observe that the type annotation `stack : list d` actually hides a coercion, and is actually `stack : list (diagram_sort d)`.

3.2 Structural duality

As briefly alluded to in the conclusion of our previous article [18], it is possible to prove a duality theorem at the meta-level of the deep-embedded first-order language. The variant presented below is updated to the current, bundled, representation of diagram types, and to a slightly different, albeit equivalent, definition of models.[1] For any formula `f : formula`, we define its dual formula `formula_dual f` by structural induction on its argument `f`, dualizing all the quivers involved in `f`. For the sake of readability, the code uses a few notations and coercions:

```
(* We fix a type for diagrams until the end of the section. *)
Variable d : Type.

(* A few local notations to ease reading *)
Local Notation model := diagram_package.
Local Notation model_dual := dual_diagram_Pack.

(* Coercion "conflating" a model with carrier type d and its associated
diagram_package. *)
Local Coercion diagram_type_of := (@diagram_type_Build d).
```

Any model of diagrams `M : model d` has a dual `model_dual M : model d`, obtained from `M` by keeping the same data, but dualizing its quiver and reversing its path relation. We can now prove the property `formula_eval_duality`, stated in Listing 2, characterizing the evaluation of this dual formula in the dual of a diagram. Observe that type `d` is the common carrier type of `M` and `model_dual M`, which allows for the same context `ctx` to be used in both sides of the equivalence.

As a corollary, if a formula holds for all diagrams in a certain diagram type, then so does its dual. The variant `duality_theorem_with_theory` is equally direct but slightly more interesting, as dependent type `P` shall be used to describe a specific class of models, e.g., models of a given theory, provided that their description is “auto-dual”, cf. Listing 2.

[1] Both files, the one attached to [18], and the one updated to fit the new definition of models, can be found in the folder `duality_theorem`.

```
Theorem formula_eval_duality (M : model d) (ctx : list d) (f : formula) :
  @formula_eval (model_dual M) ctx (formula_dual f) <-> @formula_eval M ctx f.

(* If a formula is valid in every model, then so is the dual formula *)
Corollary duality_theorem (ctx : list d) (f : formula) :
  (forall M : model d, @formula_eval M ctx f) ->
  forall M : model d, @formula_eval M ctx (formula_dual f).

(* Relativized variant to a specific class P of models *)
Corollary duality_theorem_with_theory (ctx : list d) (f : formula) (P : model d -> Prop) :
  (forall M, P M -> P (model_dual M)) -> (forall M, P M -> @formula_eval M ctx f)
  -> forall M, P M -> @formula_eval M ctx (formula_dual f).
```

Listing 2 Duality theorems.

3.3 Proofs

The deep-embedded level also features a data-structure `valid_proof` for (deep-embedded) proofs of deep-embedded formulas, and implements a checker `check_proof` for these proofs. A correctness theorem ensures that for any well-formed deep-embedded formula `f : formula`, a positive answer of the proof checker entails the provability of the interpretation of `f` in any model `d`:

```
Theorem check_proof_valid (d : diagram_type) (f : formula) (pf : valid_proof) :
  formula_wf [::] f -> check_proof f pf = true -> formula_eval d [::] f.
```

The deep-embedded level also features a data-structure `sequent`, used to reify a proof in progress, and a type `tactic` for actions making progress in a proof:

```
Inductive sequent := sequent_Build {
  context : list quiver;
  premises : list formula;
  goal : formula; }.

Definition tactic := sequent -> option sequent.
```

Note that there is only one goal attached to a sequent, as formulas are disjunction-free. To a sequent with context `[:: Q_1 ; ... ; Q_n ]`, premises `[:: H_1 ; ... ; H_m ]` and goal `G`, one can associate the following term in type `formula`:

```
Forall Q_1 ( ... ( Forall Q_n (
  Imply   (And H_1 ( ... (And H_m Ftrue) ... ))   G )) ... )
```

The sequent is well-formed if the corresponding formula is a well-formed formula. In the other direction, to any `f : formula`, one can associate the sequent with goal `f` and with empty context and no premise. A tactic τ is *valid* when for any well-formed sequent s, if $\tau\, s$ is some s', then s' is also well-formed and the evaluation of s' implies the evaluation of s. In this context, a *valid proof* is just a list of valid tactics.

To check that a valid proof actually provides a proof of some given formula f, one can perform the following steps. First, compute the sequent associated to the formula. Then, for each tactic in the proof, apply the tactic. If at some point the tactic returns `None` the proof is not correct. Otherwise, check that the goal of the final sequent is `FTrue`. In this case, the evaluation of f is valid. In the implementation, the verification is performed by the function `check_proof`, and the conclusion is proven in Theorem `check_proof_valid`.

The next step is to implement a set of *useful* tactics, and to prove that they are valid. Note that in this text, tactics refer to commands operating on deep-embedded proofs, and not on actual Coq goals. Basic tactics like introduction and elimination rules are available, but also for more involved tactics like the `Rewrite` tactic, or the `Comauto` tactic, for automating the proof of commerge problems. Starting from the rules of the proof system, more complex tactics provide relevant combinations of the existing building blocks, so as to considerably reduce the size of the proofs. Note that, in theory, the validity of certain tactics may depend on assumptions from the model used for interpreting formulas.

Here is for instance the statement of the formula corresponding to Lemma 1:

```
Definition compQ := quiver_Build 3 [:: (0,1);(0,2);(1,2)]. (* This is ◺ *)

(* when the quiver of a formula have no isolated vertex, formula_fill_vertices
allows for a shorter description of sub-quivers, only by the arcs they select. *)
Definition mono_monomPF : formula :=
formula_fill_vertices [::] (
Forall compQ
  (Commute $0 -=> monoF App (Restr {sA [:: 1]} $0) -=> monoF App (Restr {sA [:: 0]} $0))).
```

The reified proof of this statement is detaild in Section 5.

3.4 Duality for reified proofs

In fact, duality arguments are currently implemented by instrumenting proofs so as to check that they are amenable to duality arguments, which is more convenient in practice than the structural argument described in Section 3.2. Indeed, this avoids the need to prove that theories are self-dual. We thus gather a tactic τ and its dual tactic τ^* such that for any sequent s, the dual of τs is equal to τ^* applied to the dual of s. Very often, a tactic and its dual are just identical. A second theorem `duality_theorem`, not to be confused with that of Section 3.2, ensures that, indeed, if every tactic of a proof of some formula comes with such a dual tactic, then the proof obtained by taking the dual tactics will be a proof of the dual formula.[2]

```
(* Biproofs are pairs of proofs of same size *)
Structure biproof := biproof_Build {
  biproof_primal : proof;
  biproof_dual : proof;
  biproof_eq_size : size biproof_primal == size biproof_dual; }.

Theorem duality_theorem f (bpf : biproof) :
(* assuming that the tactics in the pair of proofs bpf are pairwise dual *)
  biproofD bpf ->
(* then the primal proof proves a formula iff the dual proof proves its dual *)
  check_proof (formula_dual f) (biproof_dual bpf) =
  check_proof f (biproof_primal bpf).
```

Each of the tactics currently implemented has a dual tactic. A duality argument can for instance be used to prove the dual statement to Lemma 1, and we provide the corresponding deep-embedded formula.[3]

▶ **Lemma 8.** *Let $\mathcal{C}$ be a category. For any morphism f and g such that $g \circ f \in \mathrm{Hom}(\mathcal{C})$, if $g \circ f$ is an epimorphism, then so is g.*

Proof. From Lemma 1, by duality. ◀

[2] See file `diagram_chasing/FanL.v`
[3] See file `tests_and_examples/mono_monom.v`.

```
Definition epiF :=
  Lambda_arc [:: mapQD]
  (Forall monoQD (
    EqD (Restr {sA [:: 3]} $0) $1
    -=> Commute (Restr {sA [:: 0 ; 2 ; 3]} $0)
    -=> Commute (Restr {sA [:: 1 ; 2 ; 3]} $0)
    -=> Commute (Restr {sA [:: 0 ; 1]} $0))).

Definition epi_mepiPF : formula :=
formula_fill_vertices [::] (
Forall compQD (
  Commute $0 -=> epiF App (Restr {sA [:: 1]} $0) -=> epiF App (Restr {sA [:: 0]} $0))).
```

which is similar to term `monoF` and formula `mono_monomPF` except that quivers `mapQ`, `monoQ` and `compQ` have been dualized, respectively into `mapQD`, `monoQD` and `compQD`. Observe that instead of being explicitly written, definitions `epiF` and `epi_mepiPF` can also be *computed*, respectively from `monoF` and `mono_monomPF`, by applying the function `dual_formula`. The reified proof of `epi_mepiPF` takes a single tactic, which dualizes the proof of the statement on monomorphisms.

4 Automating commutativity proofs

In order to apply most lemmas obtained by diagram chasing, one first has to prove that a certain diagram is commutative. For instance, the so-called *five lemma*, which allows to prove that some map is an isomorphism, requires to have a commutative diagram over the quiver of Figure 2. Yet proving that such a diagram is commutative by checking one equality per bipaths in this diagram would be excessively tedious. Commutativity of a larger diagram is typically obtained from the commutativity of certain sub-diagrams, and the proof of this implication is often little detailed, or not at all. For instance, in the case of Figure 2, it actually suffices to check four equalities, one for each sub-square of the quiver. More precisely, each pair of paths going from the top-left corner to the bottom-right corner of a square must correspond to equal morphisms. Once these four equalities has been proven, we infer that each square commutes.[4]

$$\begin{array}{ccccccccc} A & \longrightarrow & B & \longrightarrow & C & \longrightarrow & D & \longrightarrow & E \\ \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow \\ A' & \longrightarrow & B' & \longrightarrow & C' & \longrightarrow & D' & \longrightarrow & E' \end{array}$$

Figure 2 The five-lemma diagram.

In section 4.1, we describe an algorithm for deciding the commerge problem for diagrams with acyclic underlying quivers, so as for instance to automate the proof that the diagram of Figure 2 commutes as soon as the aforementioned four bipaths commute. In section 4.2, we describe a heuristic for discovering a collection of sub-diagrams whose commutativity entails that of a larger one.

[4] See file `diagram_chasing/Bipath.v` for a formal proof that the commutativity of a square follows from the equality of the mentioned two morphisms.

4.1 Decision procedure for the commerge problem

In [18], we provided a pen-and-paper proof of the decidability of the commerge problem for diagrams with acyclic underlying quiver, and the undecidability of its generalization to possibly cyclic underlying quivers. In this section, we describe the more practical algorithm implemented by the tactic `Comauto`. In particular, this implementation comes with a formal proof of correctness, which is the main ingredient in the validity proof of the corresponding tactic.

The algorithm operates on an acyclic quiver $\mathcal{Q}$ and a finite collection $\mathcal{Q}_1, \ldots, \mathcal{Q}_k$ of sub-quivers of $\mathcal{Q}$, representing the commutativity assumptions. Denote by l the union $\bigcup_i \mathcal{BP}_{\mathcal{Q}_i}$, slightly abusing notations by denoting $\mathcal{Q}_i$ the quiver induced by the corresponding sub-quiver. The algortithm checks whether the smallest path relation $c\ell_{\mathcal{Q}}(l)$ induced by l is full, that is $c\ell_{\mathcal{Q}}(l) = \mathcal{BP}_{\mathcal{Q}}$. Without loss of generality, we can assume that the acyclic quiver $\mathcal{Q}$ is topologically sorted in reverse order, that is, that any path in $\mathcal{Q}$ follows a sequence of vertices with decreasing labels. The implementation actually performs a topological sort of the quiver[5] and updates the representation of $c\ell_{\mathcal{Q}}(l)$ accordingly.

For any given vertices u and v in $V_{\mathcal{Q}}$, we introduce $\mathcal{A}_{u,v} \subset A_{\mathcal{Q}}$ the set of incident arrows of u starting a path to v in $\mathcal{Q}$ and $\bar{\mathcal{A}}_{v,u} \subset A_{\mathcal{Q}}$ that of incident arrows of v ending a path from u in $\mathcal{Q}$:

$$\mathcal{A}_{u,v} \triangleq \{e \in A_{\mathcal{Q}} \mid \exists p,\ e{\cdot}p \text{ is a path from } u \text{ to } v\}$$
$$\bar{\mathcal{A}}_{v,u} \triangleq \{e \in A_{\mathcal{Q}} \mid \exists p,\ p{\cdot}e \text{ is a path to } v \text{ from } u\}$$

We define $\mathcal{G}_{u,v}$ the multigraph whose vertices are the elements of $\mathcal{E}_{u,v} \triangleq \mathcal{A}_{u,v} \cup \bar{\mathcal{A}}_{v,u}$. An arc of $\mathcal{G}_{u,v}$ relates vertices $e_1, e_2 \in \mathcal{E}_{u,v}$ when:

- either there is a path in $\mathcal{Q}$ from u to v containing both e_1 and e_2;
- or there exists i and p_1, p_2 paths in $\mathcal{Q}_i$ from u to v such that e_1 (resp. e_2) appears in p_1 (resp. p_2).

We moreover introduce the binary relation $R_{u,v}$, on the elements of $\mathcal{E}$: for any $e_1, e_2 \in \mathcal{E}$, $R_{u,v}(e_1, e_2)$ holds if and only if any path p_1 from u to v containing e_1 is related by $c\ell_{\mathcal{Q}}(l)$ to any path p_2 from u to v containing e_2. If $R_{u,v}$ is full, then $c\ell_{\mathcal{Q}}(l)$ contains all the pairs of paths from u to v.

▶ **Lemma 9.** *For any vertices $u, v \in V_{\mathcal{Q}}$, if $\mathcal{G}_{u,v}$ is connected, then $R_{u,v}$ is full.*

The decision procedure constructs the multigraphs $\mathcal{G}_{u,v}$ for any pair of vertices, and checks that they are all connected. If so, then $c\ell_{\mathcal{Q}}(l)$ contains $\mathcal{BP}_{\mathcal{Q}}$.

Proof. We prove Lemma 9 for vertices respectively labelled $u+n$ and u, for a vertex $u \in V_{\mathcal{Q}}$, by induction on n. When $n = 0$, then the result holds because paths are empty.

We now assume that the result holds for any vertex and any $k < n$, and that $\mathcal{G}_{u+n,u}$ is connected. Observe first that as a consequence of the induction hypothesis, and because the graph is topologically sorted in reverse order, any two paths from $u+n$ to u sharing their initial or their final arrow are in $c\ell_{\mathcal{Q}}(l)$. We now fix $u \in V_{\mathcal{Q}}$ and $e_1, e_2 \in \mathcal{E}_{u+n,u}$ and we need to prove that $R_{u+n,u}(e_1, e_2)$. We proceed by induction on the length of a path in $\mathcal{G}_{u+n,u}$ relating e_1 and e_2. If this path is empty, then $e_1 = e_2$ is either an element of $\mathcal{A}_{u,v}$ or of $\bar{\mathcal{A}}_{v,u}$ and we can conclude using the initial observation. We now suppose that there is an arc $e \in \mathcal{E}_{u+n,u}$ and a path t in $\mathcal{G}_{u+n,u}$ such that (e_1, e) is an edge of $\mathcal{G}_{u+n,u}$ and t is a path from

[5] See `diagram_chasing/TopologicalSort.v`.

e to e_2 in $\mathcal{G}_{u+n,u}$. Let p_1 (resp. p_2) be a path in $\mathcal{Q}$ containing e_1 (resp. e_2). If there is a path p in $\mathcal{Q}$ from u to v containing both e and e_1 then p is related to p_1, by the observation, as the two paths share either their initial or their final arrow. But p is also related to p_2 by the (second) induction hypothesis as p contains e and p_2 contains e_2. The conclusion follows by transitivity of the path relation. Now suppose that e and e_1 respectively belong to q and q_1, paths from $u+n$ to u in some $\mathcal{Q}_i$. Paths q_1 and q are related, by definition of $c\ell_{\mathcal{Q}}(l)$. But q and p_2 are also related by the (second) induction hypothesis, as they respectively contain e and e_2. The conclusion follows by transitivity. ◀

4.2 Finding sufficient commutativity conditions

In fact, one can even use the computer to guess a sufficient list of equalities entailing that a certain diagram commutes, instead of providing it explicitly by hand. In this section, we explain how to do so in practice. The corresponding algorithm has been implemented under the name *comcut*. Although not strictly needed for the purpose of producing formal proofs of commerge problems, its correctness has been proven formally.[6]

Let $\mathcal{Q}$ be a quiver with vertices $V = [n]$, for some $n \in \mathbb{N}$, and arcs $A_{\mathcal{Q}}$. Moreover, we assume that $\mathcal{Q}$ is topologically sorted in a reversed order, i.e., if an arc goes from u to v, then $u > v$. From such an input, *comcut* returns a list of bipaths $l \subseteq \mathcal{BP}_{\mathcal{Q}}$ such that $c\ell_{\mathcal{Q}}(l)$, that is the smallest path relation in $\mathcal{Q}$ containing l, is equal to $\mathcal{BP}_{\mathcal{Q}}$.

The algorithm works by induction on the size of $\mathcal{Q}$. Let $u_0 := n-1$ be the top vertex. If there is no arc whose source is u_0, then we just have to apply *comcut* to $\mathcal{Q}$ deprived from the vertex u_0 (note that u_0 cannot be a target). Otherwise, let $a_0 \in A$ be an arc with source u_0. Denote by v_0 the target of a_0. Consider the quiver $\mathcal{Q}'$ obtain from $\mathcal{Q}$ by removing the arc a_0. Applying the algorithm to $\mathcal{Q}'$, we get a list $l' \subset \mathcal{BP}_{\mathcal{Q}'}$ such that $c\ell_{\mathcal{Q}'}(l') = \mathcal{BP}_{\mathcal{Q}'}$. We complete this list to a list l as follows. Let $\widetilde{W} := \mathrm{acc}(u_0) \cap \mathrm{acc}(v_0)$ be the set of vertices accessible both from u_0 and from v_0 in $\mathcal{Q}'$. Set

$$W := \{w \in \widetilde{W} \mid \forall w' \in \widetilde{W} \setminus \{w\}, w \notin \mathrm{acc}(w')\}.$$

For each $w \in W$, select a path p_w from u_0 to w in $\mathcal{Q}'$ and a path q_w from v_0 to w in $\mathcal{Q}'$. Set $l := l' \cup \{(p_w, a_0 \cdot q_w) \mid w \in W\}$.

▶ **Proposition 10.** *The list l constructed above verifies $c\ell_{\mathcal{Q}}(l) = \mathcal{BP}_{\mathcal{Q}}$.*

Before proving the proposition, let us quickly explain how to get an effective implementation from the above description. To be able to compute accessibility and to reconstruct the different paths, one computes a square matrix indexed by vertices whose (u, v) entry is either empty if there is no nontrivial path from u to v, or contains an arc a such that there is a path from u to v which starts by a. To update such a matrix for the quiver $\mathcal{Q}'$ into a matrix corresponding to the quiver $\mathcal{Q}$, it suffices to set a_0 to all the entries of the form (u_0, v) for $v \in \mathrm{acc}(v_0)$. The rest of the algorithm is easy to write down.

Proof. Let p, q be paths from u to v in $\mathcal{Q}$. Note that the arc a can only appear as the first element of p and q. If a does not appear in p nor in q, or if it appears in both, then (p, q) was already in $c\ell_{\mathcal{Q}}(l')$. Hence, up to symmetry, it remains the case where $u = u_0$, p belongs to $\mathcal{Q}'$ and $q = a_0 \cdot q'$ with q' a path of $\mathcal{Q}'$. By definition of $\widetilde{W}$, $v \in \widetilde{W}$. Let $w \in W$ such that v is accessible from w. Let r be a path from w to v (cf. Figure 3). Then, $(p, p_w \cdot r)$ and $(q_w \cdot r, q')$

[6] See the folder `comcut`.

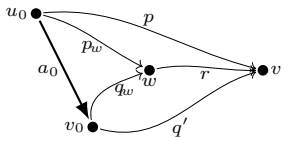


Figure 3 Decomposition of the relation between two paths thanks to an element $w \in W$.

both belong to $\mathcal{BP}_{\mathcal{Q}'}$, and $(p_w, a_0 \cdot q_w)$ belongs to l. Hence we get the following sequence of relations in $c\ell_{\mathcal{Q}}(l)$.

$$p \ \sim \ p_w \cdot r \ \sim \ a_0 \cdot q_w \cdot r \ \sim \ a_0 \cdot q' = q,$$

which proves the proposition. ◀

▶ Remark 11. Note that it may happen that l is not minimal, see Figure 4 for a counterexample.

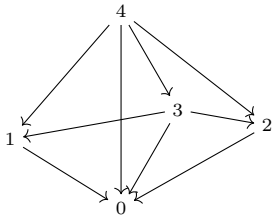


Figure 4 Counterexample to the minimality of the comcut algorithm. Before adding the arc $(4, 3)$, we need four equalities to ensure the commutativity. Adding $(4, 3)$ forces to add two more relations, but one of the previous relations becomes useless.

5 Formal proof of Lemma 1

In this section, we explain in detail the formal proof of Lemma 1 obatined using our framework[7]. This proof actually requires diagrams to be instances of a structure called `category_diagram_type`. This structure describe models which verify some compatibility conditions that we do not precise, and three more axioms. The first axiom is the existence of the composition:

$$\text{CompE:} \qquad \forall \bullet \to \bullet\, x, y, \quad \text{restr}_{\bullet \to \bullet}(x) \approx \text{restr}_{\bullet \to \bullet}(y) \ \to \ \exists \bullet \to \bullet\, z, \ \text{Comp}(x, y, z). \tag{1}$$

The last two axioms correspond to the existence and uniqueness of pushout of diagrams. Intuitively, if two diagrams coincide on a sub-quiver, we can "glue" them along this sub-quiver. For an example, see Step 2 in the proof below. See also [18] for more details on the models of these axioms.

The proof is given in Listing 3. In order to improve readability, we replaced quivers and sub-quivers by drawings. The heart of the proof is contained in `mono_monom_pf`. Before describing it in detail, we comment the rest of the code. Tactic `validify` is a custom Coq tactic which triggers Coq's unification hints [1, 24] to infer a valid proof from the proof

[7] See `mono_monom_pf` in file `tests_and_examples/mono_monom.v`.

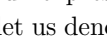
```
Definition mono_monom_pf : proof := [::
  IntroAll;
  Merge 2;
  ApplyEFT Comp [:: aT (Restr  (Var 2))];
  Merge 7;
  Comauto 4  ;
  Comauto 4  ;
  Comauto 4  ;
  Comauto 4  ;
  ApplyEFT (EqD_refl  ) [:: aT (Restr  (Var 4))];
  ApplyFT [:: aT (Restr  (Var 4)); aP 15; aP 14; aP 13] 1;
  ExactN 16
].

Definition mono_monom_vpf (diagram : category_diagram_type) : valid_proof diagram.
Proof. validify mono_monom_pf. Defined.

Lemma mono_monom (diagram : category_diagram_type) :
  @formula_eval diagram [::] mono_monomPF.
Proof.
  apply: (check_proof_valid (pf := mono_monom_vpf diagram)) ; first by [].
  vm_compute ; done.
Qed.

Definition epi_mepi_vpf (diagram : category_diagram_type) : valid_proof diagram.
Proof. dualify mono_monom_pf. Defined.

Lemma epic_mepic (diagram : category_diagram_type) :
  @formula_eval diagram [::] epi_mepiPF.
Proof.
  apply: (check_proof_valid (pf := epi_mepi_vpf diagram)) ; first by [].
  vm_compute ; done.
Qed.
```

Listing 3 A formal proof of Lemmas 1 and 8.

`mono_monom_pf`. This can be achieved because every tactic we use in our proof is canonically associated to a valid tactic. In the same way, the `dualify` Coq tactic infers dual valid proofs. Then Lemmas 1 and 8 can be proven by applying `check_proof_valid` with the corresponding valid proof. Observe that we make use of the `vm_compute` reduction machine: the default one used by Coq's type-checker is not efficient enough for executing the commerge algorithm on this nature of problems.

We now detail the list of the tactics that, when applied to the initial sequent with goal `mono_monomPF`, empty context and no premise, return a sequent of goal `FTrue`.

1. First, the tactic `IntroAll` acts as the Coq tactic `intros`: it repeats the following modifications until this is not possible anymore. If the goal is of shape `Forall Q f`, then the new goal will be `f`, while the quiver `Q` is added to the context. If the goal is of shape `f1 -=> f2`, then the new goal will be `f2`, and the formula `f1` is added to the premises. The tactic also destructs conjunctions and existential quantifiers in the new premises.
   Let us detail the sequent obtained after applying this first tactic in our case. The first quiver added to the context is  . We denote by D_0 the associated variable. Next, the two formulas commute(D_0) and Mono(restr  (D_0)) are added to the premises and are

denoted by H_0 and H_1 respectively. At this point, the goal is Mono(restr  (D_0)), whose full expression is written in Example 7. Then, the quiver  is added to the context; let us denote by D_1 the associated variable. Finally, the three formulas restr  $(D_1) \approx$ restr  (D_0), commute(restr (D_1)) and commute(restr (D_1)) are added to the premises, and the final goal is commute(restr (D_1)).

2. From the premise restr $(D_1) \approx$ restr (D_0), we can derive the existence of another variable D_2, of sort which verifies

$$\text{restr}\ \text{}\ (D_2) \approx D_1 \qquad \text{and} \qquad \text{restr}\ \text{}\ (D_2) \approx D_0.$$

The tactic `Merge` adds the new variable D_2 and the two above equalities to the sequent. Its validity is proven using the axiom of existence of pushouts of diagrams that comes from the `category_diagram_type` structure. In addition, the tactic substitutes all the occurences of D_0 and D_1 in the current sequent (including the goal) to restrictions of D_2. Thus, D_0 and D_1 can be forgotten until the end of the proof.

3. Following the diagrammatic proof of Lemma 1, we need to introduce the composition of the two arrows corresponding to the sub-quiver . It suffices to specialize the axiom of composition with these morphisms. The tactic `ApplyEFT` aims to specialize a formula, here the formula `Comp` corresponding to CompE in (1), to certain arguments, before simplifying the premises (in particular by eliminating conjunctions and existential quantifiers). After applying this tactic, we get a new variable of sort , say D_3, and a new premise: restr $(D_2) \approx$ restr (D_3).

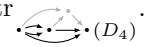
4. Once again, thanks to this premise, we can use the `Merge` tactic to merge D_2 and D_3 and to gather all the data in one variable, say D_4, of sort .

5-8. At this point of the proof, the goal is: commute(restr (D_4)). The premises contain the commutativity of the restrictions of D_4 to the following sub-quivers: , , and . In order to conclude, we have to use the premise expressing that restr (D_4) is a monomorphism, together with the commutativity of restr (D_4) and of restr (D_4). To obtain the latter, we shall use the tactic `Comauto`.

This tactic has two arguments: a variable D of sort Q, for some quiver Q, and a sub-quiver Q_0 of Q. Its purpose is to add the premise commute(restr$_{Q_0}(D)$) to the sequent after checking its correctness. It relies on the algorithm described in Section 4.1 which allows us to get the commutativity of a diagram from the commutativity of some of its subdigrams. The tactic looks for every premise of the form commute(restr$_{Q'}(D)$) for some sub-quiver Q' of Q (actually the premise might contain a composition of several restrictions) and deduces from it that restr$_{Q_0}(D)$ restricted to $Q_0 \cap Q'$ commutes. It then tries to deduce the commutativity of restr$_{Q_0}(D)$ by applying the decision procedure of the commerge problem.

From this description of the tactic, we see that we cannot directly get the commutativity of restr (D_4) and of restr (D_4). This is why we begin with checking the commutativity of restr (D_4) and of restr (D_4) thanks to two calls of the tactic `Comauto`. Then, two more calls of `Comauto` suffice to get the commutativity of restr (D_4) and of restr (D_4).

9-12. The final step of the proof consists in using the premise H_1, that is, the fact that restr (restr (restr (D_4))) is a monomorphism. This long list of restrictions

has been automatically produced by the `Merge` tactics. Following the notations of Example 7, we will apply this premise to $w \coloneqq \mathrm{restr}$  (D_4). The first thing we need to check is

$$\mathrm{restr}\ (\mathrm{restr}\ (D_4)) \approx \mathrm{restr}\ (\mathrm{restr}\ (\mathrm{restr}\ (D_4))).$$

Fortunately, the tactic `ApplyFT` automatically put the formula H_1 into a normal form, that is, each term is simplified as a single restriction of a variable to a certain sub-quiver. Hence, after normalization, the above formula just follows from the reflexivity of equality. This formula is added to premises using the tactic `ApplyEFT` on the reflexivity axiom. All the assumptions to complete the proof are in the premises, and the tactic `ApplyFT` specializes the premise $\mathrm{Mono}(\mathrm{restr}\ (D_4))$ to the right arguments. Finally, the tactic `ExactN` ends the proof, acting just as the standard `exact` Coq tactic.

6 Conclusion

We have described a first step towards the implementation of a generic library for writing reliable categorical diagrammatic proofs, available online [10]. Such a library can serve two purposes. The first one is to assist mathematician authors in writing reliable proofs, the other is to provide the mandatory infrastructure for expanding the existing corpus of formalized category theory, but also of formalized algebraic topology, and homological algebra. As expressed by the Mathlib community [15], the lack for such a tool is major showstopper for the latter. However, the current state of the present library arguably only provides a low level language for categorical statements and the next steps should enrich the collection of formula combinators, e.g. for limits, pullbacks, etc. as well as the gallery of diagram models, in particular for existing Coq formal libraries of category theory [3, 8, 27, 26]. In parallel, we would develop a interactive graphical interface so as to make the tools more user-friendly.

Independence from any library of category theory is achieved by hosting a dedicated proof system inside that of a proof assistant, Coq in this case, following the classic formalization technique of *deep-embedding* [4]. Dependent tuples allow for a structural duality property for this language. We are not aware of any comparable formal-proof-producing automation tactics for proving the commutativity of diagrams, nor for performing duality arguments. However, some existing libraries of formalized category theory, notably Mathlib [6], for the Lean proof assistant, and Unimath [26], a Coq library for univalent mathematics, provide some tools to ease proofs by diagram chasing, either with brute-force rewrite-based tactics, or with a graphical editor for generating proof scripts [13]. Gross *et al.*'s experience report [9] advocates the use of definitional equality for duality arguments, although without employing deep embeddings or quivers nor discussing diagrammatic reasoning. The other experience reports we are aware of on formalizing category theory, e.g., Carette and Hu's one in Agda [12] or Jacobs and Timany's one in Coq [25], do not include any specific support for diagrammatic proofs either. We refer the interested reader to the later article for a survey of existing libraries of formalized category theory, which remains quite relevant for the purpose of this discussion. A notable more recent endeavor is the Mathlib chapter on category theory. The later serves as a basis for Himmel's formalization of abelian categories in Lean [11], including proofs of the five lemma and of the snake lemma, and proof (semi-)automation tied to this specific formalization. Duality arguments are not addressed. Also in the Mathlib ecosystem, Monbru [21] also discusses algorithmic issues related to the automation of diagram chases, and provides incomplete heuristics for generating automatically proofs expressed in a pseudo-language.

Other computer-aided tools exist for diagrammatic categorical reasoning, with a specific emphasis on the graphical interface. Notably, the accomplished Globular/homotopy.io proof assistant [2, 7] stems from similar concerns about the reliability of diagrammatic reasoning, but for higher category theory. It is geared towards graphical representation rather than formal verification and implements various efficient algorithms for constructing and comparing diagrams in higher categories. Barras and Chabassier have designed a graphical interface for diagrammatic proofs which also provides a graphical interface for generating Coq proof scripts of string diagrams, and visualizing Coq goals as diagrams [5]. But up to our knowledge, this tool does not include any specific automation.

References

1 Andrea Asperti, Wilmer Ricciotti, Claudio Sacerdoti Coen, and Enrico Tassi. Hints in unification. In *TPHOLs*, volume 5674 of *Lecture Notes in Computer Science*, pages 84–98. Springer, 2009. doi:10.1007/978-3-642-03359-9_8.

2 Krzysztof Bar, Aleks Kissinger, and Jamie Vicary. Globular: an online proof assistant for higher-dimensional rewriting. *Log. Methods Comput. Sci.*, 14(1), 2018. doi:10.23638/LMCS-14(1:8)2018.

3 Andrej Bauer, Jason Gross, Peter LeFanu Lumsdaine, Michael Shulman, Matthieu Sozeau, and Bas Spitters. The HoTT library: a formalization of homotopy type theory in coq. In *CPP*, pages 164–172. ACM, 2017. doi:10.1145/3018610.3018615.

4 Richard J. Boulton, Andrew D. Gordon, Michael J. C. Gordon, John Harrison, John Herbert, and John Van Tassel. Experience with embedding hardware description languages in HOL. In *TPCD*, volume A-10 of *IFIP Transactions*, pages 129–156. North-Holland, 1992.

5 Luc Chabassier and Bruno Barras. A graphical interface for diagrammatic proofs in proof assistants. Contributed talks in the 29th International Conference on Types for Proofs and Programs (TYPES 2023), 2023. URL: https://types2023.webs.upv.es/TYPES2023.pdf.

6 The Mathlib Community. The lean mathematical library. In *CPP*, pages 367–381. ACM, 2020. doi:10.1145/3372885.3373824.

7 Nathan Corbyn, Lukas Heidemann, Nick Hu, Chiara Sarti, Calin Tataru, and Jamie Vicary. homotopy.io: a proof assistant for finitely-presented globular n-categories. *CoRR*, abs/2402.13179, 2024. doi:10.48550/arXiv.2402.13179.

8 Burak Ekici and Cezary Kaliszyk. Mac Lane's Comparison Theorem for the Kleisli Construction Formalized in Coq. *Math. Comput. Sci.*, 14(3):533–549, 2020. doi:10.1007/s11786-020-00450-8.

9 Jason Gross, Adam Chlipala, and David I. Spivak. Experience implementing a performant category-theory library in coq. In *ITP*, volume 8558 of *Lecture Notes in Computer Science*, pages 275–291. Springer, 2014. doi:10.1007/978-3-319-08970-6_18.

10 Benoît Guillemet, Assia Mahboubi, and Matthieu Piquerez. coq-diagram-chasing. https://gitlab.inria.fr/mpiquere/coq-diagram-chasing, 2024.

11 Markus Himmel. Diagram chasing in interactive theorem proving. Bachelorarbeit. Karlsruher Institut für Technologie, 2020. URL: https://pp.ipd.kit.edu/uploads/publikationen/himmel20bachelorarbeit.pdf.

12 Jason Z. S. Hu and Jacques Carette. Formalizing category theory in agda. In *CPP*, pages 327–342. ACM, 2021. doi:10.1145/3437992.3439922.

13 Ambroise Lafont. A categorical diagram editor to help formalising commutation proofs. https://amblafont.github.io/graph-editor/index.html, 2024. Short paper presented in the Journées Francophones des Langages Applicatifs.

14 F. William Lawvere and Stephen H. Schanuel. *Conceptual mathematics. A first introduction to categories*. Cambridge: Cambridge University Press, 2nd ed. edition, 2009.

15 leanprover-community/mathlib. Condensed mathematics/snake lemma. URL: https://leanprover-community.github.io/archive/stream/267928-condensed-mathematics/topic/snake.20lemma.html, 2021.

16 Saunders Mac Lane. *Homology*. Class. Math. Berlin: Springer-Verlag, reprint of the 3rd corr. print. 1975 edition, 1995.

17 Saunders Mac Lane. *Categories for the working mathematician*, volume 5 of *Grad. Texts Math.* New York, NY: Springer, 2nd ed edition, 1998.

18 Assia Mahboubi and Matthieu Piquerez. A first order theory of diagram chasing. In *CSL*, volume 288 of *LIPIcs*, pages 38:1–38:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.CSL.2024.38.

19 Assia Mahboubi and Enrico Tassi. *Mathematical Components*. Zenodo, January 2021. doi:10.5281/zenodo.4457887.

20 J. Peter May. *A concise course in algebraic topology*. Chicago, IL: University of Chicago Press, 1999.

21 Yannis Monbru. Towards automatic diagram chasing. M1 report. École Normale Supérieure Paris-Saclay, 2022. URL: https://github.com/ymonbru/Diagram-chasing/blob/main/MONBRU_Yannis_Rapport.pdf.

22 Matthieu Piquerez. *Tropical Hodge theory and applications*. PhD thesis, Institut Polytechnique de Paris, November 2021. URL: https://theses.hal.science/tel-03499730#.

23 Emily Riehl. *Category Theory in Context*. Dover Publications, 2017. URL: https://math.jhu.edu/~eriehl/context.pdf.

24 The Coq Development Team. The coq proof assistant, June 2023. doi:10.5281/zenodo.8161141.

25 Amin Timany and Bart Jacobs. Category theory in coq 8.5. In *FSCD*, volume 52 of *LIPIcs*, pages 30:1–30:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.FSCD.2016.30.

26 Vladimir Voevodsky, Benedikt Ahrens, Daniel Grayson, et al. Unimath – A computer-checked library of univalent mathematics. available at http://unimath.org. doi:10.5281/zenodo.10849216.

27 John Wiegley. Category theory 1.0.0. https://github.com/jwiegley/category-theory/, 2022.

Mechanized Subject Expansion in Uniform Intersection Types for Perpetual Reductions

Andrej Dudenhefner
TU Dortmund University, Germany

Daniele Pautasso
University of Turin, Italy

Abstract

We provide a new, purely syntactical proof of strong normalization for the simply typed λ-calculus. The result relies on a novel proof of the equivalence between typability in the simple type system and typability in the uniform intersection type system (a restriction of the non-idempotent intersection type system). For formal verification, the equivalence is mechanized using the Coq proof assistant.

In the present work, strong normalization of a given simply typed term M is shown in four steps. First, M is reduced to a normal form N via a suitable reduction strategy with a decreasing measure. Second, a uniform intersection type for the normal form N is inferred. Third, a uniform intersection type for M is constructed iteratively via subject expansion. Fourth, strong normalization of M is shown by induction on the size of the type derivation.

A supplementary contribution is a family of perpetual reduction strategies, i.e. strategies which preserve infinite reduction paths. This family allows for subject expansion in the intersection type systems of interest, and contains a reduction strategy with a decreasing measure in the simple type system. A notable member of this family is Barendregt's F_∞ reduction strategy.

2012 ACM Subject Classification Theory of computation → Type theory

Keywords and phrases lambda-calculus, simple types, intersection types, strong normalization, mechanization, perpetual reductions

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.8

Supplementary Material
Software (Source Code): `https://github.com/tudo-seal/uniform-intersection`
archived at `swh:1:dir:724ed6c1181e635ae0d7992cbe344798d0e7457b`

Acknowledgements The authors are grateful to Simona Ronchi Della Rocca for many insightful discussions, and to the anonymous referees for their careful reading and suggestions.

1 Introduction

Simple Types and Strong Normalization. Strong normalization (SN) of the simply typed λ-calculus (STLC) is arguably one of the cornerstones of Type Theory. Many proofs of this fundamental property have been proposed during the past decades; some are achieved by semantical means, while others adopt syntactic (inductive) approaches. On the semantical side, many SN results for typed calculi, including Gödel's system T, system F, and STLC, are obtained using *reducibility models*: these are essentially variations of the classical method due to Tait [28], or of the subsequent refinements based on Girard's reducibility candidates [15, 29, 13]. Due to their general nature, such methods often do not take into account specificities of the problem at hand, which could entail more direct and instructive SN proofs.

An alternative line of work [20, 19, 25, 26, 32, 1] focuses instead on a fine-grained analysis of combinatorial properties of term rewriting and type assignment systems. Such syntactical approaches do not always scale to more expressive calculi, but they provide insights on the

reduction process, allowing for *decreasing measures* for typed terms. The proposal at hand is a further step in this direction: our goal is to provide an accessible inductive proof of SN for STLC, capitalizing on results about intersection type systems.

Uniform Intersection Types. Pioneered by Coppo and Dezani in the late '70s [5, 6, 7], *intersection type systems* can assign to terms more than one type: writing $M : A \cap B$ intuitively means that the term M is assigned an *intersection* of the types A and B. The intersection connective $\cap$ can be understood as a notation for a *set* of types; if the idempotency of $\cap$ is dropped, i.e. $A \cap A \neq A$, intersection becomes a notation for a *multiset* of types, yielding the so called *non-idempotent* intersection types (also known as *quantitative* types). As their name suggests, non-idempotent type systems have been extensively used to study quantitative properties of programs, such as the number of reduction steps needed to reach a normal form [14, 10, 4]. It is well known that intersection type systems characterize various forms of termination, i.e. a term can be assigned an intersection type if and only if it is (strongly) normalizing [24, 8]; consequently, the type inference problem for such systems is inherently undecidable. An exception to this general rule is obtained by considering *uniform* intersection types, a particular restriction of non-idempotent intersection types; indeed, it is possible to design a *decidable* system assigning a quantitative type to all and only the simply typable terms [23]. The correspondence between the simple and uniform systems was originally established via a type inference algorithm for the uniform case: however, even if the procedure shares many similarities with the classical unification algorithm, the proof of termination in the quantitative setting relies precisely on strong normalization of simply typed terms. This makes such an approach not viable for the purposes of the present work.

Related work and contributions of this paper. Even when restricted to work adopting a syntax-oriented viewpoint, the literature about SN of STLC is quite extensive. We cannot hope to provide a detailed account here: we just point out some of the most influential contributions exploring ways to infer strong normalization of one notion of reduction from weak normalization of a finer notion of reduction, possibly after performing a translation into a suitable calculus. Some seminal ideas can be tracked back to Nederpelt [20] and Klop [19]; their techniques can be directly applied to STLC, and have been (more or less implicitly) the starting point for a number of remarkable investigations [25, 26, 32]. One of the most delicate aspects in relating weak and strong normalization is the treatment of term-erasing reductions (see [16] for a study of the connections between different proof techniques and translations into the λI-calculus, where no erasure can occur): Nederpelt and Klop's idea is to retain the subterm that would have been discarded; another solution is to delay the erasing steps as much as possible. In this sense, of great inspiration has been the research carried out by Kfoury and Wells [18], from which we borrow the notion of γ-reduction (see also [17] for a brief comparison with other methods, most notably one by de Groote [11]). Lastly, we mention recent work [1], which suggests that the search for decreasing measures for STLC is far from over.

Our contributions to this line of research are three-fold. First, the present work shows that proving SN of STLC boils down to proving that all simply typable terms are uniformly typable. This correspondence, whose novel proof is of some interest in itself, greatly simplifies the arguments commonly used when going from weak to strong normalization. In particular, we do not need to show that normalization w.r.t. the newly introduced notions of reduction implies β-strong normalization: this result comes for free thanks to the properties of quantitative type systems.

The second contribution is the identification of a family of perpetual reduction strategies, that is, strategies that diverge whenever possible [31]. We do so by carefully designing reductions for which it is easy to obtain inductive proofs of subject expansion in intersection type systems. Remarkably, this family contains Barendregt's F_∞ strategy, which was used to show that all strongly normalizing terms can be typed by a *rigid intersection* (i.e. not enjoying associativity, commutativity nor idempotency) type system without nullary intersection [22]; our approach may also be understood as an extension of such a proof method.

Third, all new results we present are mechanically verified: indeed, being fully constructive, our technique is well-suited to be formalized in a proof assistant. Experience tells that the syntactical study of term rewriting and quantitative type systems, when done exclusively by hand, is particularly error-prone; we believe that, by getting rid of this eventuality, the mechanization constitutes a valuable addition.

Paper organization. The present work is structured as follows:

Section 2: Preliminaries on the λ-calculus, definition of the $I\gamma K'$-reduction.

Section 3: Measure-based weak $I\gamma K'$-normalization of simply typed terms (Theorem 15).

Section 4: Uniform intersection type system (Definition 21), uniform typability of β-normal forms (Lemma 24), subject expansion for the $I\gamma K'$-reduction (Lemma 27, Lemma 30, and Lemma 31), and consequently, strong normalization of simply typed terms (Theorem 34).

Section 5: Family of perpetual reduction strategies (Corollary 40) via generalization of subject expansion properties to the non-idempotent intersection type system (Theorem 39).

Section 6: Overview over the mechanization in the Coq proof assistant.

Section 7: Concluding remarks.

2 Preliminaries on Calculus and Reductions

Let us fix the basic notation for the remainder of the paper (following standard literature [2]). *Terms* of the λ-*calculus* are generated by the following grammar:

$$M, N \ ::= \ x \ \mid \ \lambda x.M \ \mid \ MN$$

where x ranges over a countable set of term variables. $\mathsf{FV}(M)$ denotes the set of free variables of the term M.

The β-*reduction*, denoted $\to_\beta$, is the contextual closure of the rule

$$(\lambda x.M)N \mapsto_\beta M[N/x]$$

where $M[N/x]$ denotes the capture-free substitution of x by N in M. A term of shape $(\lambda x.M)N$ is called a β-redex. Such β-redexes are partitioned into I-redexes and K-redexes, depending on whether the variable x occurs free in M or not; this distinction is of central importance in the subsequent sections. Formally, the reductions $\to_I$ and $\to_K$ are, respectively, the contextual closure of the rules:

$$(\lambda x.M)N \mapsto_I M[N/x] \text{ if } x \in \mathsf{FV}(M) \qquad (\lambda x.M)N \mapsto_K M \text{ if } x \notin \mathsf{FV}(M)$$

Given a binary relation $\to_r$, we denote its reflexive, transitive closure with $\to_r^*$. Given two binary relations $\to_{r_1}$ and $\to_{r_2}$, we write $\to_{r_1 r_2}$ for $\to_{r_1} \cup \to_{r_2}$; clearly $\to_\beta = \to_{IK} = \to_I \cup \to_K$. A term is in r-*normal form* when it does not contain any r-redex; it is (weakly) r-*normalizing* if it can be reduced to a term in r-normal form by a r-reduction sequence; it is *strongly* r-*normalizing* if every r-reduction sequence starting from it eventually stops.

We introduce the following reduction $\to_{K'}$ (Definition 1) as a refinement of the K-reduction. The reduction $\to_{K'}$ can only contract K-redexes in specific positions, motivated by two goals. First, K'-expansion should preserve strong normalization, allowing for subject expansion in suitable intersection type systems. Second, a term in IK'-normal form should be β-normal, otherwise IK'-reduction would get stuck on β-reducible terms.

▶ **Definition 1** ($\to_{K'}$).
1. *If N is in β-normal form and $x \notin \mathsf{FV}(M)$, then $(\lambda x.M)N \to_{K'} M$.*
2. *If $M \to_{K'} N$, then $\lambda x.M \to_{K'} \lambda x.N$.*
3. *If $N_1 \to_{K'} N_2$, then $xM_1 \dots M_n N_1 \to_{K'} xM_1 \dots M_n N_2$, where $n \geq 0$.*
4. *If $N_1 \to_{K'} N_2$ and $x \notin \mathsf{FV}(M)$, then $(\lambda x.M)N_1 \to_{K'} (\lambda x.M)N_2$.*
5. *If $M_1 M_2 \to_{K'} M_3$, then $(M_1 M_2)N \to_{K'} M_3 N$.*

The following Example 2, Remark 3, and Remark 4 give insight into the relationship between $\to_{K'}$ and perpetual reduction strategies.

▶ **Example 2.** Consider the term $\omega = \lambda z.zz$ and the term $M = (\lambda x.(\lambda y.x)(xx))\omega$, which is not strongly β-normalizing because of the sequence $M \to_\beta (\lambda y.\omega)(\omega\omega) \to_\beta (\lambda y.\omega)(\omega\omega) \to_\beta \dots$ The term M K-reduces to $(\lambda x.x)\omega$ for which the infinite β-reduction sequence is lost. However, the term M cannot be K'-reduced.

▶ Remark 3. By proving subject expansion for K'-reduction in suitable intersection type systems, we obtain that any IK'-reduction strategy is perpetual, thanks to the characterization of strongly normalizing terms via typability (see for example [2, Theorem 17.2.15] for the idempotent case, and [4, Corollary 8.4] for the non-idempotent one).

▶ Remark 4. The IK'-reduction admits the F_∞ perpetual reduction strategy [31, Definition 3.21].

The following Lemma 5 shows that $\to_{IK'}$ cannot get stuck on $\to_\beta$-reducible terms.

▶ **Lemma 5.** *If $M \to_\beta N$ then there exists N' such that $M \to_{IK'} N'$.*

Proof. If M contains at least one I-redex, the result is immediate. Otherwise, M can contain K-redexes only, and we proceed by induction on M. Case $M = x$ vacuously holds, while case $M = \lambda x.P$ follows by inductive hypothesis and point (2) of Definition 1. Lastly, consider the case $M = PQ$ is an application.

- If $P = x$ then by inductive hypothesis and point (3) of Definition 1 we can reduce Q.
- If $P = \lambda x.S$ then PQ is a K-redex and we distinguish two subcases:
  - if Q is in normal form, by point (1) of Definition 1 we can reduce PQ;
  - if Q is not in normal form, by inductive hypothesis and point (4) of Definition 1 we can reduce Q.
- If $P = S_1 S_2$ we distinguish two subcases:
  - if P is in normal form, then Q is not in normal form; by inductive hypothesis and point (3) of Definition 1 we can reduce Q;
  - if P is not in normal form, by inductive hypothesis and point (5) of Definition 1 we can reduce P. ◀

The last essential ingredient of the present work is the commutation rule

$$(\lambda x.\lambda y.M)N \mapsto_\gamma \lambda y.(\lambda x.M)N,$$

whose contextual closure we denote $\to_\gamma$ [18, Definition 3.1].

A γ-reduction step can be understood as the combination of one β-reduction step and one β-expansion step, rearranging the structure of a term, without altering its "meaning". The idea is that γ-reduction can be used to postpone K-redexes, possibly exposing I-redexes, as illustrated by the following Example 6.

▶ **Example 6.** Consider the term $M = (\lambda y.\lambda x.x)wz$. Using the IK'-reduction, the term M is reduced to a normal form as follows: $M \to_{K'} (\lambda x.x)z \to_I z$. Using γ-reduction, the K'-redex can be postponed: $M \to_\gamma (\lambda x.(\lambda y.x)w)z \to_I (\lambda y.z)w \to_{K'} z$.

3 Simple Types and a Decreasing Measure

In this section we show that simply typed terms are $I\gamma K'$-normalizing. Similarly to the approach by Kfoury and Wells [18], we proceed in two steps. First, given a simply typed term we construct an $I\gamma$-normal form using a decreasing measure for rightmost $I\gamma$-redex contraction. The decreasing measure is different from the one by Kfoury and Wells [18, Lemma 4.3], and does not require a specific interleaving of I-reductions and γ-reductions (cf. $\star$-reduction [18, Definition 3.8]). Second, we iteratively contract K'-redexes in order to construct a β-normal form.

Let us briefly recollect the simple type assignment system.

▶ **Definition 7** (Simple Types). *The set $\mathcal{T}_{\mathcal{S}}$ of* simple types *is defined by the grammar $\sigma, \tau ::= a \mid \sigma \to \tau$, where a ranges over a countable set of type variables.*

A *type environment* is a finite, functional set of pairs $x : \sigma$, where x is a term variable and σ a simple type; environments are ranged over by $\Gamma, \Delta, \Phi, \Psi$. If $x : \sigma \in \Gamma$, then $\Gamma(x) = \sigma$; the domain of an environment Γ is $\mathsf{dom}(\Gamma) = \{x \mid x : A \in \Gamma\}$; Γ and Δ *agree*, written $\Gamma \smile \Delta$, if $\Gamma(x) = \Delta(x)$ for all $x \in \mathsf{dom}(\Gamma) \cap \mathsf{dom}(\Delta)$. The writing Γ, Δ is short for $\Gamma \cup \Delta$ in case $\mathsf{dom}(\Gamma) \cap \mathsf{dom}(\Delta) = \emptyset$; the writing $\Gamma, x : \sigma$ is a special case of such notation when $x \notin \mathsf{dom}(\Gamma)$.

▶ **Definition 8** (Simple Type Assignment System). *The simple type assignment system $\mathcal{S}$ derives judgments of shape $\Gamma \vdash M : \sigma$, where Γ is an environment, M is a term, and σ is a simple type. The rules of $\mathcal{S}$ are as follows:*

$$\frac{}{\Gamma, x : \sigma \vdash x : \sigma}\ (\mathsf{var}) \qquad \frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x.M : \sigma \to \tau}\ (\to_{\mathsf{I}})$$

$$\frac{\Gamma \vdash M : \sigma \to \tau \quad \Delta \vdash N : \sigma \quad \Gamma \smile \Delta}{\Gamma \cup \Delta \vdash MN : \tau}\ (\to_{\mathsf{E}})$$

Type derivations are ranged over by Π, Σ, Θ. We often write $\Gamma \vdash M : \sigma$ as a shorthand for the existence of a derivation proving $\Gamma \vdash M : \sigma$, and when we want to name a particular derivation with such conclusion we write $\Pi \triangleright \Gamma \vdash M : \sigma$. Additionally, each subterm N of M is associated with exactly one judgment $\Delta \vdash N : \tau$ in Π, and we may annotate the assigned simple type τ onto N, by writing N^τ.

▶ **Lemma 9.** *If $\Gamma \vdash M : \sigma$ and $M \to_{I\gamma K'} N$, then $\Gamma \vdash N : \sigma$.*

Proof. For $\to_{IK'} \subseteq \to_\beta$ the property follows by subject reduction ([2, Proposition 1.2.6]). For $\to_\gamma$ the property follows from the generation lemma ([2, Proposition 1.2.3]). ◀

The *measure* $\mathsf{meas}(M)$ of a simply typed term M is a multiset of pairs (m, n) of natural numbers, where m carries information about the size of involved types, and n is the depth of the redex w.r.t. abstraction. Multisets are written in square brackets, and multiset union is denoted $\uplus$, taking multiplicities into account.

Each pair in the multiset $\mathsf{meas}(M)$ is associated with one $I\gamma$-redex in M, and is called *rank* of the redex. If a subterm $(\lambda x.\lambda y.M)^{\sigma_1\to\sigma_2\to\tau}N$ is both a γ-redex and an I-redex, then its rank is that of the I-redex. Correspondingly, we prefer I-redexes in case of ambiguity for the rightmost redex.

▶ **Definition 10** (Measure of a simply typed λ-term)**.**

- *The* size *of a simple type is defined as:*

$$\mathsf{size}(a) = 1 \qquad \mathsf{size}(\sigma\to\tau) = 1 + \mathsf{size}(\sigma) + \mathsf{size}(\tau)$$

- *The* measure *of a simply typed term with corresponding simple type annotations is:*

$$\begin{array}{lcll}
\mathsf{meas}(x) & = & [\,] & \\
\mathsf{meas}(\lambda x.M) & = & [(k,n+1) \mid (k,n) \in \mathsf{meas}(M)] & \\
\mathsf{meas}((\lambda x.M)^{\sigma\to\tau}N) & = & \mathsf{meas}(\lambda x.M) \uplus \mathsf{meas}(N) \uplus & \\
 & & [(\mathsf{size}(\sigma\to\tau),0)] & \text{if } x \in \mathsf{FV}(M) \\
\mathsf{meas}((\lambda x.\lambda y.M)^{\sigma_1\to\sigma_2\to\tau}N) & = & \mathsf{meas}(\lambda x.\lambda y.M) \uplus \mathsf{meas}(N) \uplus & \\
 & & [(1+\mathsf{size}(\sigma_2\to\tau),0)] & \text{if } x \notin \mathsf{FV}(M) \\
\mathsf{meas}(MN) & = & \mathsf{meas}(M) \uplus \mathsf{meas}(N) & \text{otherwise}
\end{array}$$

Pairs are ordered lexicographically and multisets are ordered by the multiset ordering [12]. The following Lemma 11 shows that rightmost $I\gamma$-redex contraction is measure-decreasing.

▶ **Lemma 11.** *If M, N are simply typed terms such that $M \to_{I\gamma} N$ by contracting the rightmost (in the textual presentation) redex, then $\mathsf{meas}(N) < \mathsf{meas}(M)$ by the multiset order.*

Proof. Since we reduce the rightmost $I\gamma$-redex, no $I\gamma$-redex is duplicated. Therefore, it suffices to show that each new $I\gamma$-redex created by the reduction is of smaller rank than that of the contracted redex.

First, we consider cases in which the contraction of an I-redex (as the rightmost $I\gamma$-redex) may create new redexes [21].

- Case $(\lambda x.M)^{\sigma\to\tau}N \to_I M[N/x]$ and there is a subterm xP of M.
  - In case $N = \lambda y.Q$ and $y \in \mathsf{FV}(Q)$ we have that $\sigma = \sigma_1 \to \sigma_2$ for some σ_1, σ_2 and $\mathsf{size}(\sigma_1\to\sigma_2) < \mathsf{size}(\sigma\to\tau)$. Therefore, the created I-redex $(\lambda y.Q)^{\sigma_1\to\sigma_2}P$ is of smaller rank (regardless of its depth in M).
  - In case $N = \lambda y.\lambda z.Q$ and $y \notin \mathsf{FV}(Q)$ we have that $\sigma = \sigma_1 \to \sigma_2 \to \sigma_3$ for some $\sigma_1, \sigma_2, \sigma_3$ and $1 + \mathsf{size}(\sigma_2\to\sigma_3) < \mathsf{size}(\sigma\to\tau)$. Therefore, the created γ-redex $(\lambda y.\lambda z.Q)^{\sigma_1\to\sigma_2\to\sigma_3}P$ is of smaller rank.
- Case $(\lambda x.x)^{(\sigma\to\tau)\to(\sigma\to\tau)}NP \to_I NP$.
  - In case $N = \lambda y.Q$ and $y \in \mathsf{FV}(Q)$ the created I-redex $(\lambda y.Q)^{\sigma\to\tau}P$ is of smaller rank.
  - In case $N = \lambda y.\lambda z.Q$ and $y \notin \mathsf{FV}(Q)$ the created γ-redex $(\lambda y.\lambda z.Q)^{\sigma\to\tau}P$ is of smaller rank.
- Case $(\lambda x.\lambda y.M)^{\sigma_1\to\sigma_2\to\tau}NP \to_I (\lambda y.M[N/x])^{\sigma_2\to\tau}P$ and $y \in \mathsf{FV}(M)$.
  Since $\mathsf{size}(\sigma_2\to\tau) < \mathsf{size}(\sigma_1\to\sigma_2\to\tau)$, the created I-redex is of smaller rank.
- Case $(\lambda x.\lambda y.\lambda z.M)^{\sigma_1\to\sigma_2\to\sigma_3\to\tau}NP \to_I (\lambda y.\lambda z.M[N/x])^{\sigma_2\to\sigma_3\to\tau}P$ and $y \notin \mathsf{FV}(M)$.
  Since $1 + \mathsf{size}(\sigma_3\to\tau) < \mathsf{size}(\sigma_1\to\sigma_2\to\sigma_3\to\tau)$, the created γ-redex is of smaller rank.

Second, we consider cases in which the contraction of a γ-redex (as the rightmost $I\gamma$-redex) may create new redexes. The contracted γ-redex cannot be an (otherwise preferred) I-redex.

- Case $(\lambda x.\lambda y.\lambda z.M)^{\sigma_1\to\sigma_2\to\sigma_3\to\tau}N \to_\gamma \lambda y.(\lambda x.\lambda z.M)^{\sigma_1\to\sigma_3\to\tau}N$ such that $x \notin \mathsf{FV}(M)$.
  Since $1 + \mathsf{size}(\sigma_3\to\tau) < 1 + \mathsf{size}(\sigma_2\to\sigma_3\to\tau)$, the created γ-redex is of smaller rank.

- Case $(\lambda x.\lambda y.M)^{\sigma_1\to\sigma_2\to\tau}NP \to_\gamma (\lambda y.(\lambda x.M)N)^{\sigma_2\to\tau}P$ such that $x \notin \mathsf{FV}(M)$, $y \in \mathsf{FV}(M)$. Since $\mathsf{size}(\sigma_2 \to \tau) < 1 + \mathsf{size}(\sigma_2 \to \tau)$, the created I-redex is of smaller rank. The subterm $(\lambda x.M)N$ is not an I-redex because $x \notin \mathsf{FV}(M)$; the case in which $(\lambda x.M)N$ is a new γ-redex is already treated above.
- Case $(\lambda x.(\lambda y.\lambda z.M)^{\sigma_2\to\sigma_3\to\tau}N)P \to_\gamma (\lambda x.\lambda z.(\lambda y.M)N)^{\sigma_1\to\sigma_3\to\tau}P$ such that $y \notin \mathsf{FV}(M)$, $x \notin \mathsf{FV}(M) \cup \mathsf{FV}(N)$. The contracted γ-redex is of rank $(1+\mathsf{size}(\sigma_3 \to \tau), n+1)$ for some n, and the created γ-redex is of smaller rank $(1 + \mathsf{size}(\sigma_3 \to \tau), n)$. ◀

▶ Remark 12. The last case in the above proof of Lemma 11 explains the definition of the rank of a γ-redex. In particular, the rank of a γ-redex *does not* depend on the type of the first abstracted variable, and it *does* depend on the depth of the γ-redex w.r.t. abstraction.

The following Lemma 13 is a weaker variant of the strong normalization property of $I\gamma$-normal forms [18, Lemma 3.10].

▶ **Lemma 13.** *If M, N are terms such that $M \to_{K'} N$ by contracting the rightmost K'-redex and M is $I\gamma$-normal, then N is $I\gamma$-normal and N has fewer β-redexes than M.*

Proof. Consider the K'-redex $(\lambda x.P)Q$ in M. The term P cannot be an abstraction, otherwise $(\lambda x.P)Q$ would have been a γ-redex, contradicting the assumption that M is in $I\gamma$-normal form. Therefore, contracting $(\lambda x.P)Q$ to P introduces neither β-redexes nor γ-redexes. ◀

The combination of Lemma 11 and Lemma 13 provides a normalization strategy (Theorem 15) for simply typed terms, which is illustrated in the following Example 14.

▶ **Example 14.** Consider the annotated term $M = (\lambda y.\lambda x.x)^{b\to a\to a}wz$ from Example 6, which can be a assigned the type a in the type environment $\{z : a, w : b\}$. The term M with measure $\mathsf{meas}(M) = [(4, 0)]$ can be reduced to a normal form as follows:

$$(\lambda y.\lambda x.x)^{b\to a\to a}wz \quad \to_\gamma \quad (\lambda x.(\lambda y.x)^{b\to a}w)^{a\to a}z \quad \to_I \quad (\lambda y.z)^{b\to a}w \quad \to_{K'} \quad z$$

The initial γ-reduction postpones the top-level K-redex, exposes an I-redex, and decreases the measure to $\mathsf{meas}((\lambda x.(\lambda y.x)^{b\to a}w)^{a\to a}z) = [(3, 0)]$. The subsequent I-reduction leaves only K-redexes and the measure decreases to $\mathsf{meas}((\lambda y.z)^{b\to a}w) = [\,]$. Finally, the term is normalized using the K'-reduction.

▶ **Theorem 15.** *Given a simply typed term M, there exists a β-normal form N such that $M \to^*_{I\gamma K'} N$.*

Proof. By induction on $\mathsf{meas}(M)$, repeatedly contracting the rightmost $I\gamma$-redex we obtain a $I\gamma$-normal form P by Lemma 11. By induction on the number of K-redexes in P, repeatedly contracting the rightmost K'-redex (we cannot get stuck by Lemma 5) we obtain a β-normal form by Lemma 13. ◀

▶ Remark 16. Of course, we can show Theorem 15 using a strong normalization argument for the simply typed λ-calculus together with Lemma 5. However, it is methodologically intriguing to utilize measure-based weak normalization in pursuit of typability in a non-idempotent intersection type system (which constitutes a strong normalization proof).

▶ Remark 17. The advantage of the additional γ-reduction is apparent in the design of a type-based decreasing measure for IK'-normalization. Consider the term $(\lambda x.uxx)^{\sigma_1\to\tau_1}(vM)$ where $M = (\lambda y.\lambda z.N)^{\sigma_2\to\sigma_2\to\tau_2}ww$ such that $y \notin \mathsf{FV}(N)$ and $z \in \mathsf{FV}(N)$. Since $x \in \mathsf{FV}(uxx)$ we cannot contract the K-redex occurring in M using K'-reduction. However, contracting the I-redex $(\lambda x.uxx)^{\sigma_1\to\tau_1}(vM) \to_I u(vM)(vM)$ duplicates M. Contracting each redex

copy in M results in two new I-redexes $(\lambda z.N)^{\sigma_2\to\tau_2}w$. In sum, contracting an I-redex with the associated type $\sigma_1 \to \tau_1$ results in two copies of an I-redex with the associated (arbitrary large) type $\sigma_2 \to \tau_2$. For reference, the F_∞ [31, Definition 3.21] perpetual reduction strategy also involves the described duplication. In comparison, by γ-reduction $(\lambda y.\lambda z.N)ww \to_\gamma (\lambda z.(\lambda y.N)w)w$ the K-redex $(\lambda y.N)w$ is delayed and the I-redex is exposed for contraction (without duplication).

4 Uniform Intersection Types

As previously mentioned, uniform intersection types are a restriction of non-idempotent intersection types based on the notion of uniform multiset. From now on we will frequently use indexed types, where indexes are natural numbers: the symbols I, J will denote sets of indexes. For the sake of simplicity we adopt the same notation for types and uniform types (resp. multisets and uniform multisets), as the intended meaning can be easily inferred from the context.

▶ **Definition 18.**

- *Non-idempotent intersection types ($\mathcal{T}_\mathcal{I}$) are inductively defined by the grammar:*

$$\begin{array}{llll} \textsc{Intersection Types} & A,B,C & ::= & a \mid \mu \to A \\ \textsc{Multisets} & \mu,\nu & ::= & [A_1,\dots,A_n] \quad (n \geq 1) \end{array}$$

- *Equivalence relation $\sim$ on intersection types (uniformity):*

$a \sim a$ *for all type variables* a

$\mu \to A \sim \nu \to B$ *iff* $\mu \sim \nu$ *and* $A \sim B$

$[A_i]_{i\in I} \sim [B_j]_{j\in J}$ *iff* $A_i \sim B_j$ *for all* $i \in I, j \in J$

- *Uniform intersection types ($\mathcal{T}_\mathcal{U}$) are inductively defined by the grammar:*

$$\begin{array}{llll} \textsc{Unif. Int. Types} & A,B,C & ::= & a \mid \mu \to A \\ \textsc{Unif. Multisets} & \mu,\nu & ::= & [A_1,\dots,A_n] \quad \forall i,j \in \{1,\dots,n\}.A_i \sim A_j \end{array}$$

Remark that in both grammars the empty multiset is *not* allowed. The types $A = [[a,a,a] \to b, [a] \to b] \to c$ and $B = [[a,a] \to b] \to c$ are uniform, whereas $[a,[a] \to b] \to c$ is not; moreover, observe that $A \sim B$. The intuition is that uniform types are the quantitative version of simple types; two uniform types are equivalent if they correspond to the same underlying simple type. Given a uniform type, the underlying simple type can be easily recovered by means of a translation that "forgets" non-idempotency.

▶ **Definition 19.** *The* collapse translation $\mathsf{c} : \mathcal{T}_\mathcal{I} \to \mathcal{T}_\mathcal{S}$ *is a partial function recursively defined as:*

$$\begin{array}{lcll} \mathsf{c}(a) & = & a & \\ \mathsf{c}([A_1,\dots,A_n] \to B) & = & \sigma \to \mathsf{c}(B) & \text{if } \mathsf{c}(A_1) = \dots = \mathsf{c}(A_n) = \sigma \end{array}$$

▶ **Lemma 20.**

- *The collapse translation $\mathsf{c}(\cdot)$ is a total function on $\mathcal{T}_\mathcal{U}$.*
- *If $A, B \in \mathcal{T}_\mathcal{U}$, then $A \sim B$ if and only if $\mathsf{c}(A) = \mathsf{c}(B)$.*

Proof. The two points are proved at the same time, by mutual induction on the structure of uniform types and the definition of collapse translation. ◀

We are now ready to introduce the uniform intersection type assignment system $\mathcal{U}$. In this setting, we say that a type environment Γ is *uniform* if it associates each term variable to a *uniform multiset*. If Γ and Δ are two such environments, $\Gamma \sim \Delta$ means that $\Gamma(x) \sim \Delta(x)$ for all $x \in \mathsf{dom}(\Gamma) \cap \mathsf{dom}(\Delta)$. The union of environments is defined as $(\Gamma_0 \uplus \Gamma_1)(x) = \Gamma_0(x) \uplus \Gamma_1(x)$ if $x \in \mathsf{dom}(\Gamma_0) \cap \mathsf{dom}(\Gamma_1)$, while $(\Gamma_0 \uplus \Gamma_1)(x) = \Gamma_i(x)$ if $x \in \mathsf{dom}(\Gamma_i)$ and $x \notin \mathsf{dom}(\Gamma_{1-i})$.

▶ **Definition 21.** *The uniform intersection type assignment system $\mathcal{U}$, assigning types in $\mathcal{T}_{\mathcal{U}} \subset \mathcal{T}_{\mathcal{I}}$ to terms, consists of the following rules:*

$$\frac{A \in \mu \quad \Gamma \text{ and } \mu \text{ uniform}}{\Gamma, x : \mu \vdash_{\mathtt{u}} x : A}\ (\mathsf{var}) \qquad \frac{\Gamma, x : \mu \vdash_{\mathtt{u}} M : A}{\Gamma \vdash_{\mathtt{u}} \lambda x.M : \mu \to A}\ (\to_{\mathsf{I}})$$

$$\frac{\Gamma_0 \vdash_{\mathtt{u}} M : [A_1, \dots, A_n] \to B \quad (\Gamma_i \vdash_{\mathtt{u}} N : A_i)_{1 \le i \le n} \quad \forall i, j \in \{0, \dots, n\}.\Gamma_i \sim \Gamma_j}{\biguplus_{i=0}^{n} \Gamma_i \vdash_{\mathtt{u}} MN : B}\ (\to_{\mathsf{E}})$$

The full non-idempotent intersection type assignment system, which we call system $\mathcal{I}$, is easily obtained from system $\mathcal{U}$ by removing the uniformity constraint on multisets. It is clear that each derivation in system $\mathcal{U}$ is also a valid derivation in system $\mathcal{I}$; we use the symbol $\vdash_{\mathtt{i}}$ to explicitly distinguish judgments in system $\mathcal{I}$.

The collapse translation is naturally extended to uniform multisets and uniform type environments, so that the translation of a derivation in system $\mathcal{U}$ is a derivation in system $\mathcal{S}$.

▶ **Definition 22.** *The collapse translation $\mathsf{c}(\Pi)$ of a derivation $\Pi \rhd \Gamma \vdash_{\mathtt{u}} M : A$ is the simple type derivation inductively defined as follows:*

- *If Π ends with a (var) rule, i.e. $\Pi \rhd \Gamma \vdash_{\mathtt{u}} x : A$, then $\mathsf{c}(\Pi) \rhd \mathsf{c}(\Gamma) \vdash M : \mathsf{c}(A)$.*
- *If Π ends with a $(\to_{\mathsf{I}})$ rule, i.e. has shape:*

$$\frac{\Pi_0 \rhd \Gamma, x : \mu \vdash_{\mathtt{u}} N : B}{\Pi \rhd \Gamma \vdash_{\mathtt{u}} \lambda x.N : A = \mu \to B} \quad \text{then} \quad \frac{\mathsf{c}(\Pi_0) \rhd \mathsf{c}(\Gamma), x : \mathsf{c}(\mu) \vdash N : \mathsf{c}(B)}{\mathsf{c}(\Pi) \rhd \mathsf{c}(\Gamma) \vdash \lambda x.N : \mathsf{c}(A) = \mathsf{c}(\mu) \to \mathsf{c}(B)}$$

- *If Π ends with a $(\to_{\mathsf{E}})$ rule, i.e. has shape:*

$$\frac{\Pi_0 \rhd \Gamma_0 \vdash_{\mathtt{u}} P : [B_1, \dots, B_n] \to A \qquad (\Pi_i \rhd \Gamma_i \vdash_{\mathtt{u}} Q : B_i)_{1 \le i \le n}}{\Pi \rhd \Gamma \vdash_{\mathtt{u}} PQ : A}$$

then, letting $\mathsf{c}(\Pi_i) \rhd \mathsf{c}(\Gamma_i) \vdash_{\mathtt{u}} Q : \mathsf{c}(B_i)$ and recalling that $\mathsf{c}(B_i) = \tau$ for all $1 \le i \le n$:

$$\frac{\mathsf{c}(\Pi_0) \rhd \mathsf{c}(\Gamma_0) \vdash P : \tau \to \mathsf{c}(A) \qquad (\bigcup_{i=1}^{n} \mathsf{c}(\Gamma_i)) \vdash Q : \tau}{\mathsf{c}(\Pi) \rhd \mathsf{c}(\Gamma) \vdash PQ : \mathsf{c}(A)} \qquad \text{where } \mathsf{c}(\Gamma) = \bigcup_{i=0}^{n} \mathsf{c}(\Gamma_i)$$

▶ **Lemma 23** ([23, Theorem 34])**.** *$\Pi \rhd \Gamma \vdash_{\mathtt{u}} M : A$ implies $\mathsf{c}(\Pi) \rhd \mathsf{c}(\Gamma) \vdash M : \mathsf{c}(A)$.*

Showing the converse, namely that all simply typable terms can also be assigned a uniform intersection type, is not as easy. In what follows we provide an alternative proof of this claim, adopting a dual approach w.r.t. previous work [23]: instead of reasoning about term reduction, we reason about term expansion. Additionally, strong normalization is a consequence in our case, and not a prerequisite [23, Theorem 25].

4.1 Uniform Typability of Normal Forms

As a first step, we show that system $\mathcal{U}$ can assign a uniform type to all simply typable terms in β-normal form.

▶ **Lemma 24.** *If $\Sigma \triangleright \Gamma \vdash M : \sigma$ and M is in β-normal form, then there exists $\Pi \triangleright \Gamma' \vdash_{\mathtt{u}} M : A$ such that $\mathsf{c}(\Pi) = \Sigma$.*

Proof. By induction on the term M. Recall that β-normal forms are defined by the grammar: $M, N ::= \lambda x.M \mid xM_1 \dots M_n$ where $n \geq 0$.

Case $M = x$ is immediate, and case $M = \lambda x.N$ follows by inductive hypothesis. Lastly, consider the case $M = xM_1 \dots M_n$ such that $n \geq 1$ and we have $\Sigma \triangleright \bigcup_{i=0}^{n} \Gamma_i \vdash xM_1 \dots M_n : \sigma$. Then there exist $\Sigma_0 \triangleright \Gamma_0 \vdash x : \tau_1 \to \dots \to \tau_n \to \sigma$ and $\Sigma_i \triangleright \Gamma_i \vdash M_i : \tau_i$ $(1 \leq i \leq n)$ such that $\Gamma_i \smile \Gamma_j$ for all $i, j \in \{0, \dots, n\}$. By inductive hypothesis there are $\Pi_i \triangleright \Gamma'_i \vdash_{\mathtt{u}} M_i : B_i$ such that $\mathsf{c}(\Pi_i) = \Sigma_i$, and consequently $\mathsf{c}(\Gamma'_i) = \Gamma_i$ and $\mathsf{c}(B_i) = \tau_i$ $(1 \leq i \leq n)$. Moreover, it is easy to build $\Pi_0 \triangleright \Gamma'_0 \vdash_{\mathtt{u}} x : [B_1] \to \dots \to [B_n] \to A$ such that $\mathsf{c}(\Gamma'_0) = \Gamma_0$ and $\mathsf{c}(A) = \sigma$, thus satisfying $\mathsf{c}(\Pi_0) = \Sigma_0$. Remark that

$$\mathsf{c}([B_1] \to \dots \to [B_n] \to A) = \mathsf{c}(B_1) \to \dots \to \mathsf{c}(B_n) \to \mathsf{c}(A) = \tau_1 \to \dots \to \tau_n \to \sigma$$

Since $\mathsf{c}(\Gamma'_i) = \Gamma_i$ and $\Gamma_i \smile \Gamma_j$, we know that $\Gamma'_i(y) \sim \Gamma'_j(y)$ for all $y \in \mathsf{dom}(\Gamma'_i) \cap \mathsf{dom}(\Gamma'_j)$ $(i, j \in \{0, \dots, n\})$. Therefore, we can use the various Π_i $(0 \leq i \leq n)$ to build a derivation $\Pi \triangleright \biguplus_{i=0}^{n} \Gamma'_i \vdash_{\mathtt{u}} xM_1 \dots M_n : A$ such that $\mathsf{c}(\Pi) = \Sigma$. ◀

4.2 Typability-preserving Expansions

Now that we know that simply typable β-normal forms are uniformly typable, the crucial step is showing that (simply typed) subject expansion w.r.t. $\to_I$, $\to_\gamma$ and $\to_{K'}$ preserves typability in system $\mathcal{U}$. The fact that I-expansion preserves typability in system $\mathcal{I}$ is folklore; here we need to specialize the result to the particular case of system $\mathcal{U}$.

▶ **Notation 25.** *Given $\Pi \triangleright \Gamma \vdash_{\mathtt{u}} M : A$ and $\Delta \sim \Gamma$, we write $\Pi^{(\Delta)} \triangleright \Gamma \uplus \Delta \vdash_{\mathtt{u}} M : A$ for the derivation obtained from Π by weakening.*

▶ **Notation 26.** *Let $\Sigma \triangleright \Gamma \vdash M : \sigma$. If $M \to_\beta N$, we write $\Sigma \rightsquigarrow \Sigma' \triangleright \Gamma \vdash N : \sigma$ meaning that Σ' is obtained from Σ by mimicking the β-reduction on the simple type derivation.*[1]

▶ **Lemma 27.** *Let $\Pi_N \triangleright \Gamma \vdash_{\mathtt{u}} N : A$. If $M \to_I N$ and there is $\Sigma \triangleright \Phi \vdash M : \sigma$ such that $\Sigma \rightsquigarrow \mathsf{c}(\Pi_N^{(\Delta)}) \triangleright \Phi \vdash N : \sigma$ for some $\Delta \sim \Gamma$, then $\Gamma \vdash_{\mathtt{u}} M : A$.*

Proof. By induction on the reduction context. Remark that $\Sigma \rightsquigarrow \mathsf{c}(\Pi_N^{(\Delta)}) \triangleright \mathsf{c}(\Gamma \uplus \Delta) = \Phi \vdash N : \mathsf{c}(A) = \sigma$ for some $\Delta \sim \Gamma$ implies $\mathsf{c}(\Gamma) \subseteq \Phi$. For the base case, let $M = (\lambda x.P)Q \to_I P[Q/x] = N$ and, wlog, assume $x \notin \mathsf{FV}(Q)$. We show how to build $\Pi_M \triangleright \Gamma \vdash_{\mathtt{u}} M : A$ starting from Π_N. The derivation Π_N contains a finite number of subderivations with subject Q: let them be $\Theta_i \triangleright \Gamma_i \vdash_{\mathtt{u}} Q : B_i$ $(i \in I = \{1, \dots, n\})$, and let $\Gamma = \Gamma_0 \uplus_{i \in I} \Gamma_i$. Since $\Sigma \rightsquigarrow \mathsf{c}(\Pi_N^{(\Delta)})$, it must be the case that $\mathsf{c}(B_i) = \mathsf{c}(B_j)$ for all $i, j \in I$. By Lemma 20 this implies $B_i \sim B_j$ for all $i, j \in I$, hence the multiset $[B_i]_{i \in I}$ is uniform. Substituting in Π_N each subderivation Θ_i with an axiom $x : [B_i] \vdash_{\mathtt{u}} x : B_i$ yields a derivation with conclusion $\Gamma_0, x : [B_i]_{i \in I} \vdash_{\mathtt{u}} P : A$. Therefore we can build:

$$\dfrac{\dfrac{\Gamma_0, x : [B_i]_{i \in I} \vdash_{\mathtt{u}} P : A}{\Gamma_0 \vdash_{\mathtt{u}} \lambda x.P : [B_i]_{i \in I} \to A}\,(\to_{\mathtt{I}}) \qquad (\Gamma_i \vdash_{\mathtt{u}} Q : B_i)_{i \in I}}{\Pi_M \triangleright \Gamma \vdash_{\mathtt{u}} (\lambda x.P)Q : A}\,(\to_{\mathtt{E}})$$

For the inductive step, consider the contexts in which a reduction may take place:

[1] That is: given a subderivation typing $(\lambda x.P)Q$, substitute the axioms typing x in the subderivation for P by the subderivation typing Q, so to obtain a subderivation with subject $P[Q/x]$.

- Case $M = \lambda x.P \to_I \lambda x.Q = N$. Letting $\sigma = \tau_1 \to \tau_2$ and $A = \mu \to B$, the derivations Σ and Π_N have shape:

$$\frac{\Sigma_0 \rhd \Phi, x : \tau_1 \vdash P : \tau_2}{\Sigma \rhd \Phi \vdash \lambda x.P : \tau_1 \to \tau_2}\,(\to_{\mathsf{I}}) \qquad \frac{\Pi_0 \rhd \Gamma, x : \mu \vdash_{\mathtt{u}} Q : B}{\Pi_N \rhd \Gamma \vdash_{\mathtt{u}} \lambda x.Q : \mu \to B}\,(\to_{\mathsf{I}})$$

  $\Sigma \rightsquigarrow \mathsf{c}(\Pi_N^{(\Delta)})$ means that $\Sigma_0 \rightsquigarrow \mathsf{c}(\Pi_0^{(\Delta)})$. Therefore by inductive hypothesis there exists $\Gamma, x : \mu \vdash_{\mathtt{u}} P : B$, from which one obtains $\Pi_M \rhd \Gamma \vdash_{\mathtt{u}} \lambda x.P : \mu \to B$.
- Case $M = PQ$ is an application. Let Σ be:

$$\frac{\Sigma_1 \rhd \Phi_1 \vdash P : \tau \to \sigma \qquad \Sigma_2 \rhd \Phi_2 \vdash Q : \tau}{\Sigma \rhd \Phi \vdash PQ : \sigma}\,(\to_{\mathsf{E}})$$

  First, consider the case $M = PQ \to_I PR = N$. Letting $I = \{1, \dots, n\}$, the derivation Π_N has shape:

$$\frac{\Pi_0 \rhd \Gamma_0 \vdash_{\mathtt{u}} P : [B_i]_{i \in I} \to A \qquad (\Pi_i \rhd \Gamma_i \vdash_{\mathtt{u}} R : B_i)_{i \in I}}{\Pi_N \rhd \Gamma \vdash_{\mathtt{u}} PR : A}\,(\to_{\mathsf{E}})$$

  $\Sigma \rightsquigarrow \mathsf{c}(\Pi_N^{(\Delta)})$ means that, for each $i \in I$, $\Sigma_2 \rightsquigarrow \mathsf{c}(\Pi_i^{(\Delta_i)})$ for some $\Delta_i \sim \Gamma_i$. Therefore by inductive hypothesis there exist $\Gamma_i \vdash_{\mathtt{u}} Q : B_i$ $(i \in I)$, and we conclude.
  Now consider the case $M = PQ \to_{K'} RQ = N$. The derivation Π_N has shape:

$$\frac{\Pi_0 \rhd \Gamma_0 \vdash_{\mathtt{u}} R : [B_i]_{i \in I} \to A \qquad (\Pi_i \rhd \Gamma_i \vdash_{\mathtt{u}} Q : B_i)_{i \in I}}{\Pi_N \rhd \Gamma \vdash_{\mathtt{u}} RQ : A}\,(\to_{\mathsf{E}})$$

  Similarly to the previous scenario, $\Sigma \rightsquigarrow \mathsf{c}(\Pi_N^{(\Delta)})$ implies $\Sigma_1 \rightsquigarrow \mathsf{c}(\Pi_0^{(\Delta_0)})$ for some $\Delta_0 \sim \Gamma_0$. Therefore by inductive hypothesis there is $\Gamma_0 \vdash_{\mathtt{u}} P : [B_i]_{i \in I} \to A$, and we conclude. ◀

The following Example 28 illustrates subject expansion w.r.t. the I-reduction.

▶ **Example 28.** Consider $M = (\lambda x.yxx)P \to_I yPP = N$. Assuming N is uniformly typable, let Π_N be the uniform type derivation:

$$\dfrac{\dfrac{\Gamma_0 \vdash_{\mathtt{u}} y : [B_1] \to [B_2] \to A \qquad \Gamma_1 \vdash_{\mathtt{u}} P : B_1}{\Gamma_0 \uplus \Gamma_1 \vdash_{\mathtt{u}} yP : [B_2] \to A} \qquad \Gamma_2 \vdash_{\mathtt{u}} P : B_2}{\Pi_N \rhd \Gamma \vdash_{\mathtt{u}} yPP : A}$$

Its collapse translation is the simple type derivation $\mathsf{c}(\Pi_N)$:

$$\dfrac{\dfrac{\mathsf{c}(\Gamma_0) \vdash y : \mathsf{c}(B_1) \to \mathsf{c}(B_2) \to \mathsf{c}(A) \qquad \mathsf{c}(\Gamma_1) \vdash P : \mathsf{c}(B_1)}{\mathsf{c}(\Gamma_0 \uplus \Gamma_1) \vdash yP : \mathsf{c}(B_2) \to \mathsf{c}(A)} \qquad \mathsf{c}(\Gamma_2) \vdash P : \mathsf{c}(B_2)}{\mathsf{c}(\Pi_N) \rhd \mathsf{c}(\Gamma) \vdash yPP : \mathsf{c}(A)}$$

Saying that there is $\Sigma \rhd \Phi \vdash M : \sigma$ such that $\Sigma \rightsquigarrow \mathsf{c}(\Pi_N)$, implies that Σ has shape:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{c}(\Gamma_0) \vdash y : \tau \to \tau \to \mathsf{c}(A) \qquad \mathsf{c}(\Gamma_1), x : \tau \vdash x : \tau}{\mathsf{c}(\Gamma_0 \uplus \Gamma_1), x : \tau \vdash yx : \tau \to \mathsf{c}(A)} \qquad \mathsf{c}(\Gamma_2), x : \tau \vdash x : \tau}{\mathsf{c}(\Gamma), x : \tau \vdash yxx : \mathsf{c}(A)}}{\mathsf{c}(\Gamma) \vdash \lambda x.yxx : \tau \to \mathsf{c}(A)} \qquad \mathsf{c}(\Gamma_1) \cap \mathsf{c}(\Gamma_2) \vdash P : \tau}{\Sigma \rhd \mathsf{c}(\Gamma) \vdash (\lambda x.yxx)P : \mathsf{c}(A)}$$

where $\mathsf{c}(\Gamma) = \Phi$, $\mathsf{c}(A) = \sigma$, and $\mathsf{c}(B_1) = \mathsf{c}(B_2) = \tau$. In turn, this entails $B_1 \sim B_2$; therefore the multiset $[B_1, B_2]$ is uniform, and we can build the uniform derivation Π_M as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma_0 \vdash_{\mathtt{u}} y : [B_1] \to [B_2] \to A \qquad x : [B_1] \vdash_{\mathtt{u}} x : B_1}{\Gamma_0, x : [B_1] \vdash_{\mathtt{u}} yx : [B_2] \to A} \qquad x : [B_2] \vdash_{\mathtt{u}} x : B_2}{\Gamma_0, x : [B_1, B_2] \vdash_{\mathtt{u}} yxx : A}}{\Gamma_0 \vdash_{\mathtt{u}} \lambda x.yxx : [B_1, B_2] \to A} \qquad \Gamma_1 \vdash_{\mathtt{u}} P : B_1 \qquad \Gamma_2 \vdash_{\mathtt{u}} P : B_2}{\Pi_M \rhd \Gamma \vdash_{\mathtt{u}} (\lambda x.yxx)P : A}$$

It might not be obvious why one needs the condition $\Sigma \rightsquigarrow \mathsf{c}(\Pi_N^{(\Delta)})$ in the statement of Lemma 27 (and, similarly, in the later Lemma 31). Given $\Pi_N \rhd \Gamma \vdash_{\mathtt{u}} N : A$ and a term M such that $M \to_I N$, the reader may wonder if a weaker hypothesis, e.g. only requiring $\Sigma \rhd \mathsf{c}(\Gamma \uplus \Delta) \vdash M : \mathsf{c}(A)$, would suffice to prove $\Pi_M \rhd \Gamma \vdash_{\mathtt{u}} M : A$. The following Example 29 shows that such a formulation would not work in the inductive case, specifically when N is an application: in order to use the inductive hypothesis, one must relate the structures of Σ and Π_N.

▶ **Example 29.** Let $\Delta = \emptyset$ and consider a closed, simply typable term P such that $P \to_I Q$ (for instance, $P = (II)I \to_I II = Q$ where $I = \lambda x.x$). For $N = (\lambda z.y)Q$ let Π_N be the uniform derivation:

$$\dfrac{\dfrac{z : [B], y : [A] \vdash_{\mathtt{u}} y : A}{y : [A] \vdash_{\mathtt{u}} \lambda z.y : [B] \to A} \qquad \Pi_Q \rhd \vdash_{\mathtt{u}} Q : B}{\Pi_N \rhd y : [A] \vdash_{\mathtt{u}} (\lambda z.y)Q : A}$$

Assume there is $\Sigma \rhd y : \mathsf{c}(A) \vdash (\lambda z.y)P : \mathsf{c}(A)$. We would like to exploit the information about Σ to build a derivation $\Pi_M \rhd y : [A] \vdash_{\mathtt{u}} (\lambda z.y)P : A$; however, there is no guarantee that there exists a simple derivation $\Sigma_P \rhd \vdash P : \mathsf{c}(B)$, so we cannot use the inductive hypothesis to get $\Pi_P \rhd \vdash_{\mathtt{u}} P : B$. On the other hand, if we know $\Sigma \rightsquigarrow \mathsf{c}(\Pi_N)$, we can deduce that Σ has shape:

$$\dfrac{\dfrac{z : \mathsf{c}(B), y : \mathsf{c}(A) \vdash y : \mathsf{c}(A)}{y : \mathsf{c}(A) \vdash \lambda z.y : \mathsf{c}(B) \to \mathsf{c}(A)} \qquad \Sigma_P \rhd \vdash P : \mathsf{c}(B)}{\Sigma \rhd y : \mathsf{c}(A) \vdash (\lambda z.y)P : \mathsf{c}(A)}$$

where $\Sigma_P \rightsquigarrow \mathsf{c}(\Pi_Q)$. Thus, by inductive hypothesis there exists $\Pi_P \rhd \vdash_{\mathtt{u}} P : B$, from which it is possible to build the desired Π_M.

Proving that typability in $\mathcal{U}$ is preserved by γ-expansion is straightforward. Notice that, as opposed to Lemma 27, the term M is not explicitly required to be simply typable.

▶ **Lemma 30.** *If $\Gamma \vdash_{\mathtt{u}} N : A$ and $M \to_\gamma N$, then $\Gamma \vdash_{\mathtt{u}} M : A$.*

Proof. The proof proceeds by induction on the reduction context. For the base case, let $M = (\lambda x.\lambda y.P)Q \to_\gamma \lambda y.(\lambda x.P)Q = N$; observe that by α-conversion we can freely assume $y \notin \mathsf{FV}(Q)$. Thus, the derivation $\Pi_N \rhd \Gamma \vdash_{\mathtt{u}} N : A$ has shape:

$$\dfrac{\dfrac{\dfrac{\Delta, x : [B_i]_{i\in I}, y : [C_j]_{j\in J} \vdash_{\mathtt{u}} P : D}{\Delta, y : [C_j]_{J\in j} \vdash_{\mathtt{u}} \lambda x.P : [B_i]_{i\in I} \to D}\,(\to_{\mathsf{I}}) \qquad (\Delta_i \vdash_{\mathtt{u}} Q : B_i)_{i\in I}}{\Gamma, y : [C_j]_{J\in j} \vdash_{\mathtt{u}} (\lambda x.P)Q : D}\,(\to_{\mathsf{E}})}{\Pi_N \rhd \Gamma \vdash_{\mathtt{u}} \lambda y.(\lambda x.P)Q : [C_j]_{j\in J} \to D}\,(\to_{\mathsf{I}})$$

By rearranging the derivation rules we can easily build $\Pi_M \rhd \Gamma \vdash_{\mathtt{u}} M : A$:

$$\dfrac{\dfrac{\dfrac{\Delta, x : [B_i]_{i\in I}, y : [C_j]_{j\in J} \vdash_{\mathtt{u}} P : D}{\Delta, x : [B_i]_{i\in I} \vdash_{\mathtt{u}} \lambda y.P : [C_j]_{j\in J} \to D}\,(\to_{\mathsf{I}})}{\Delta \vdash_{\mathtt{u}} \lambda x.\lambda y.P : [B_i]_{i\in I} \to [C_j]_{j\in J} \to D}\,(\to_{\mathsf{I}}) \qquad (\Delta_i \vdash_{\mathtt{u}} Q : B_i)_{i\in I}}{\Pi_M \rhd \Gamma \vdash_{\mathtt{u}} (\lambda x.\lambda y.P)Q : [C_j]_{j\in J} \to D}\,(\to_{\mathsf{E}})$$

The inductive cases immediately follow using the inductive hypothesis. ◀

Before moving on to K'-expansion, we briefly discuss why dealing with arbitrary K-expansion would be quite problematic, even in the unrestricted system $\mathcal{I}$. Assume two derivations $\Theta \triangleright \Gamma \vdash_{\mathtt{i}} M : A$ and $\Delta \vdash_{\mathtt{i}} N : B$, where $x \notin \mathsf{FV}(M)$. It is straightforward to build $\Theta' \triangleright \Gamma \uplus \Delta \vdash_{\mathtt{i}} (\lambda x.M)N : A$, thus reversing the K-reduction step $(\lambda x.M)N \to_K M$. However, notice that in general $\Gamma \uplus \Delta$ contains bigger multisets than the ones originally found in Γ; therefore, if Θ is a subderivation of a larger derivation Π, simply replacing Θ by Θ' may not result in a correct derivation. In order to be consistent with the enlarged multisets, it may be necessary to *globally* update the structure of Π: this means introducing new subderivations and/or replicating existing ones (along with their type environments), which in turn may lead to further inconsistencies.

On the other hand, restricting the focus to K'-expansions has the great advantage of keeping the required modifications *local*, thus allowing for an elegant inductive reasoning. The proof of the following Lemma 31 clearly illustrates this point.

▶ **Lemma 31.** *Let $\Pi_N \triangleright \Gamma \vdash_{\mathtt{u}} N : A$. If $M \to_{K'} N$ and there is $\Sigma \triangleright \Phi \vdash M : \sigma$ such that $\Sigma \rightsquigarrow \mathsf{c}(\Pi_N^{(\Delta)}) \triangleright \Phi \vdash N : \sigma$ for some $\Delta \sim \Gamma$, then:*

- *if M is not an abstraction, then $\Gamma' \vdash_{\mathtt{u}} M : A$ for some Γ' such that $\mathsf{c}(\Gamma') \subseteq \Phi$;*
- *if M is an abstraction, then $\Gamma' \vdash_{\mathtt{u}} M : A'$ for some Γ' and A' such that $\mathsf{c}(\Gamma') \subseteq \Phi$ and $A' \sim A$.*

Proof. By induction on the reduction context. Remark that $\Sigma \rightsquigarrow \mathsf{c}(\Pi_N^{(\Delta)}) \triangleright \mathsf{c}(\Gamma \uplus \Delta) = \Phi \vdash N : \mathsf{c}(A) = \sigma$ for some $\Delta \sim \Gamma$ implies $\mathsf{c}(\Gamma) \subseteq \Phi$. We show how to build a derivation Π_M with the desired properties starting from Π_N. For the base case, consider $M = (\lambda x.N)P \to_{K'} N$, where P is in β-normal form. The derivation Σ has shape:

$$\dfrac{\dfrac{\Phi_1, x : \tau \vdash N : \sigma}{\Sigma_1 \triangleright \Phi_1 \vdash \lambda x.N : \tau \to \sigma}\,(\to_{\mathtt{I}}) \qquad \Sigma_2 \triangleright \Phi_2 \vdash P : \tau}{\Sigma \triangleright \Phi \vdash (\lambda x.N)P : \sigma}\,(\to_{\mathtt{E}})$$

By Lemma 24 there exists $\Pi_2 \triangleright \Gamma_2 \vdash_{\mathtt{u}} P : B$ such that $\mathsf{c}(\Pi_2) = \Sigma_2$, which entails $\mathsf{c}(\Gamma_2) = \Phi_2 \subseteq \Phi$ and $\mathsf{c}(B) = \tau$. Note that $\mathsf{c}(\Gamma) \subseteq \Phi$ guarantees $\Gamma \sim \Gamma_2$. Starting from Π_N, it is easy to exploit weakening in the axioms and obtain a derivation with conclusion $\Gamma, x : [B] \vdash_{\mathtt{u}} N : A$. Hence we can build:

$$\dfrac{\dfrac{\Gamma, x : [B] \vdash_{\mathtt{u}} N : A}{\Gamma \vdash_{\mathtt{u}} \lambda x.N : [B] \to A}\,(\to_{\mathtt{I}}) \qquad \Gamma_2 \vdash_{\mathtt{u}} P : B}{\Pi_M \triangleright \Gamma' = \Gamma \uplus \Gamma_2 \vdash_{\mathtt{u}} (\lambda x.N)P : A}\,(\to_{\mathtt{E}})$$

satisfying the requirements. Indeed, $\mathsf{c}(\Gamma) \subseteq \Phi$ and $\mathsf{c}(\Gamma_2) = \Phi_2 \subseteq \Phi$ imply $\mathsf{c}(\Gamma') \subseteq \Phi$.

For the inductive step, consider the reduction contexts in which a K'-reduction may take place:

- Case $M = \lambda x.P \to_{K'} \lambda x.Q = N$. Letting $\sigma = \tau_1 \to \tau_2$ and $A = \mu \to B$, the derivations Σ and Π_N have shape:

$$\dfrac{\Sigma_0 \triangleright \Phi, x : \tau_1 \vdash P : \tau_2}{\Sigma \triangleright \Phi \vdash \lambda x.P : \tau_1 \to \tau_2}\,(\to_{\mathtt{I}}) \qquad \dfrac{\Pi_0 \triangleright \Gamma, x : \mu \vdash_{\mathtt{u}} Q : B}{\Pi_N \triangleright \Gamma \vdash_{\mathtt{u}} \lambda x.Q : \mu \to B}\,(\to_{\mathtt{I}})$$

  $\Sigma \rightsquigarrow \mathsf{c}(\Pi_N^{(\Delta)})$ means that $\Sigma_0 \rightsquigarrow \mathsf{c}(\Pi_0^{(\Delta)})$. Therefore, if P is an abstraction, by inductive hypothesis there are Γ', μ' and B' such that $\mathsf{c}(\Gamma', x : \mu') \subseteq (\Phi, x : \tau_1)$, $B' \sim B$ and $\Gamma', x : \mu' \vdash_{\mathtt{u}} P : B'$. From this we obtain $\Pi_M \triangleright \Gamma' \vdash_{\mathtt{u}} \lambda x.P : \mu' \to B'$ satisfying the requirements. If P is not an abstraction, the reasoning is similar.

- Case $M = xM_1 \dots M_m P \to_{K'} xM_1 \dots M_m Q = N$, where $m \geq 0$. The derivation Σ has shape:

$$\frac{\Sigma_1 \rhd \Phi_1 \vdash xM_1 \dots M_m : \tau \to \sigma \qquad \Sigma_2 \rhd \Phi_2 \vdash P : \tau}{\Sigma \rhd \Phi \vdash xM_1 \dots M_m P : \sigma}\ (\to_{\mathsf{E}})$$

  Letting $I = \{1, \dots, n\}$, the derivation Π_N has shape:

$$\dfrac{\dfrac{\Psi \vdash_{\mathtt{u}} x : \mu_1 \to \ldots \to \mu_m \to [B_i]_{i\in I} \to A \qquad (\Psi_C \vdash_{\mathtt{u}} M_1 : C)_{C\in\mu_1}}{\vdots}\ (\to_{\mathsf{E}})}{\dfrac{\Pi_0 \rhd \Gamma_0 \vdash_{\mathtt{u}} xM_1 \dots M_m : [B_i]_{i\in I} \to A \qquad (\Pi_i \rhd \Gamma_i \vdash_{\mathtt{u}} Q : B_i)_{i\in I}}{\Pi_N \rhd \Gamma \vdash_{\mathtt{u}} xM_1 \dots M_m Q : A}\ (\to_{\mathsf{E}})}$$

  $\Sigma \rightsquigarrow \mathsf{c}(\Pi_N^{(\Delta)})$ means that, for each $i \in I$, $\Sigma_2 \rightsquigarrow \mathsf{c}(\Pi_i^{(\Delta_i)})$ for some $\Delta_i \sim \Gamma_i$. Therefore, if P is an abstraction, by inductive hypothesis there are $\Pi'_i \rhd \Gamma'_i \vdash_{\mathtt{u}} P : B'_i$ such that $\mathsf{c}(\Gamma'_i) \subseteq \Phi_2 \subseteq \Phi$ and $B'_i \sim B_i$ $(i \in I)$. Note that $\mathsf{c}(\Gamma'_i) \subseteq \Phi$ for all $i \in I$ guarantees $\Gamma'_i \sim \Gamma'_j$ and $\Gamma_0 \sim \Gamma'_i$ for all $i, j \in I$. Hence to obtain Π_M it suffices to replace Π_i by Π'_i, and change the type of the axiom introducing x into $\mu_1 \to \ldots \to \mu_m \to [B'_i]_{i\in I} \to A$. In case P is not an abstraction, the reasoning is similar.
- Case $M = (\lambda x.P)Q \to_{K'} (\lambda x.P)R = N$, where $x \notin \mathsf{FV}(P)$. The derivation Σ has shape:

$$\frac{\Sigma_1 \rhd \Phi_1 \vdash \lambda x.P : \tau \to \sigma \qquad \Sigma_2 \rhd \Phi_2 \vdash Q : \tau}{\Sigma \rhd \Phi \vdash (\lambda x.P)Q : \sigma}\ (\to_{\mathsf{E}})$$

  Letting $I = \{1, \dots, n\}$, the derivation Π_N has shape:

$$\frac{\Pi_0 \rhd \Gamma_0 \vdash_{\mathtt{u}} \lambda x.P : [B_i]_{i\in I} \to A \qquad (\Pi_i \rhd \Gamma_i \vdash_{\mathtt{u}} R : B_i)_{i\in I}}{\Pi_N \rhd \Gamma \vdash_{\mathtt{u}} (\lambda x.P)R : A}\ (\to_{\mathsf{E}})$$

  $\Sigma \rightsquigarrow \mathsf{c}(\Pi_N^{(\Delta)})$ means that, for each $i \in I$, $\Sigma_2 \rightsquigarrow \mathsf{c}(\Pi_i^{(\Delta_i)})$ for some $\Delta_i \sim \Gamma_i$. Therefore, if Q is an abstraction, by inductive hypothesis there exists $\Pi'_i \rhd \Gamma'_i \vdash_{\mathtt{u}} Q : B'_i$ such that $\mathsf{c}(\Gamma'_i) \subseteq \Phi_2 \subseteq \Phi$ and $B'_i \sim B_i$ $(i \in I)$. As in the previous case, from $\mathsf{c}(\Gamma'_i) \subseteq \Phi$ $(i \in I)$ we deduce $\Gamma'_i \sim \Gamma'_j$ and $\Gamma_0 \sim \Gamma'_i$ $(i, j \in I)$. Thus, in order to build Π_M, it suffices to replace Π_i by Π'_i and change the multiset associated to the dummy variable x, so that it matches the new types B'_i. The case in which Q is not an abstraction is similar.
- Case $M = (PQ)R \to_{K'} SR = N$. The derivation Σ has shape:

$$\frac{\Sigma_1 \rhd \Phi_1 \vdash PQ : \tau \to \sigma \qquad \Sigma_2 \rhd \Phi_2 \vdash R : \tau}{\Sigma \rhd \Phi \vdash (PQ)R : \sigma}\ (\to_{\mathsf{E}})$$

  Letting $I = \{1, \dots, n\}$, the derivation Π_N has shape:

$$\frac{\Pi_0 \rhd \Gamma_0 \vdash_{\mathtt{u}} S : [B_i]_{i\in I} \to A \qquad (\Pi_i \rhd \Gamma_i \vdash_{\mathtt{u}} R : B_i)_{i\in I}}{\Pi_N \rhd \Gamma \vdash_{\mathtt{u}} SR : A}\ (\to_{\mathsf{E}})$$

  $\Sigma \rightsquigarrow \mathsf{c}(\Pi_N^{(\Delta)})$ implies $\Sigma_1 \rightsquigarrow \mathsf{c}(\Pi_0^{(\Delta_0)})$ for some $\Delta_0 \sim \Gamma_0$. Therefore, as PQ is not an abstraction, by inductive hypothesis there is $\Pi'_0 \rhd \Gamma'_0 \vdash_{\mathtt{u}} PQ : [B_i]_{i\in I} \to A$ such that $\mathsf{c}(\Gamma'_0) \subseteq \Phi_1 \subseteq \Phi$. Since $\mathsf{c}(\Gamma_i) \subseteq \Phi$, it holds that $\Gamma'_0 \sim \Gamma_i$ for all $i \in I$. The derivation Π_M is obtained by replacing Π_0 by Π'_0. ◀

Finally, we have all prerequisites in order to prove that if a term is typable by $\mathcal{S}$, then it is also typable by $\mathcal{U}$.

▶ **Theorem 32.** $\Sigma \triangleright \Gamma \vdash M : \sigma$ *implies there is* $\Pi \triangleright \Gamma' \vdash_{\mathtt{u}} M : A$ *such that* $\mathsf{c}(\Pi) = \Sigma$.

Proof. Consider a $I\gamma K'$-reduction sequence s from M to a β-normal form, which exists by Theorem 15. We reason by induction on the length n of the sequence s. If $n = 0$, i.e. M is in β-normal form, the result immediately follows from Lemma 24. For the inductive part of the proof, we rely on Lemma 27 for I-reduction steps, on Lemma 30 for γ-reduction steps, and on Lemma 31 for K'-reduction steps. ◀

4.3 From Uniform Typability to Strong Normalization

It is notoriously easy to show that all terms which are typable in system $\mathcal{I}$ (and, consequently, all terms which are typable in system $\mathcal{U}$) are strongly β-normalizing. Intuitively, this is because non-idempotent intersection type systems internalize the reduction process: if M is a term containing a subterm N, a quantitative derivation for M requires (at least) as many subderivations for N as there are copies of N that can be produced during any β-reduction sequence from M to its normal form. Since all the required copies are already there to begin with, mimicking β-reduction on the quantitative derivation necessarily decreases its total size: indeed, I-reduction steps simply rearrange the derivation structure, replacing axioms by subderivations, while K-reduction steps erase subderivations. Formally, writing $\mathsf{size}(\Pi)$ for the number of rules in a derivation Π, one has that:

▶ **Theorem 33** (Weighted Subject Reduction [3, Theorem 4.2])**.** *If* $\Pi_M \triangleright \Gamma \vdash_{\mathtt{i}} M : A$ *and* $M \to_\beta N$*, then there exists* $\Pi_N \triangleright \Gamma \vdash_{\mathtt{i}} N : A$ *such that* $\mathsf{size}(\Pi_N) < \mathsf{size}(\Pi_M)$.

Finally, we can state the following.

▶ **Theorem 34.** $\Gamma \vdash M : \sigma$ *implies* M *is strongly* β*-normalizing.*

Proof. Immediate consequence of Theorem 32 and Theorem 33. ◀

5 A Family of Perpetual Reductions

This brief section shows that $I\gamma K'$-expansion also holds in the general system $\mathcal{I}$. Actually, the proof is simpler, because there are no requirements concerning simple typability. Such a result, together with the fact that all $\mathcal{I}$-typable terms are strongly normalizing, allows us to identify a family of perpetual reduction strategies.

We begin by pointing out that Lemma 24 and Lemma 27 are the system $\mathcal{U}$ counterparts of the following well-known properties of system $\mathcal{I}$:

▶ **Lemma 35.** *If* M *is in* β*-normal form, then* $\Gamma \vdash_{\mathtt{i}} M : A$.

Proof. Essentially as in Lemma 24. ◀

▶ **Lemma 36** ([4, Theorem 4.3])**.** *If* $\Gamma \vdash_{\mathtt{i}} N : A$ *and* $M \to_I N$*, then* $\Gamma \vdash_{\mathtt{i}} M : A$.

Similar considerations can be made for the other typability-preserving expansions.

▶ **Lemma 37.** *If* $\Gamma \vdash_{\mathtt{i}} N : A$ *and* $M \to_\gamma N$*, then* $\Gamma \vdash_{\mathtt{i}} M : A$.

Proof. By observing that the proof of Lemma 30 never mentions uniformity. ◀

One needs to be careful with the statement of subject expansion w.r.t. the K'-reduction (the counterpart of Lemma 31): the case in which the contracted term is an abstraction does not preserve the assigned type.

▶ **Lemma 38.** *If* $\Gamma \vdash_{\mathtt{i}} N : A$ *and* $M \to_{K'} N$*, then:*
- *if* M *is not an abstraction, then* $\Gamma' \vdash_{\mathtt{i}} M : A$ *for some* Γ'*;*
- *if* M *is an abstraction, then* $\Gamma' \vdash_{\mathtt{i}} M : A'$ *for some* Γ' *and* A'*.*

Proof. By induction on the reduction context. The proof is analogous to that of Lemma 31, without the conditions on simple typability. The base case, namely $M = (\lambda x.N)P \to_{K'} N$ where P is in β-normal form, relies on Lemma 35. ◀

We now have all the ingredients to state the following:

▶ **Theorem 39.** *If* M *is* $I\gamma K'$*-normalizing, then it is strongly* β*-normalizing.*

Proof. Lemmas 35, 36, 37 and 38 guarantee that there exists $\Gamma \vdash_{\mathtt{i}} M : A$. Then the result immediately follows from Theorem 33. ◀

▶ **Corollary 40.** *Any* $I\gamma K'$*-reduction strategy is perpetual.*

6 Mechanization

This section provides an overview over the mechanization[2] of uniform typability of simply typed terms (Theorem 32) using the Coq proof assistant [30]. The mechanization is axiom-free and spans approximately 2000 lines of code, consisting of the following four parts:
- `stlc.v` and `stlc_facts.v` contain definitions and facts (such as subject reduction and substitution lemmas) for the simple type system.
- `stlc_nf.v` proves that simply typed terms are $I\gamma K'$-normalizing (Theorem 15).
- `nitlc.v` and `nitlc_facts.v` contain definitions and facts (such as weakening) for the uniform intersection type system.
- `nitlc_typ.v` proves the equivalence between simple type typability and uniform intersection type typability (Lemma 23 and Theorem 32).

Simple types and annotated λ-terms are mechanized in `stlc.v` as `sty` and `tm` respectively. Variable binding is addressed via the unscoped de Bruijn approach [9], with infrastructure partially generated by `Autosubst 2` [27].

```
Inductive sty : Type :=
  | satom (x : nat)   (* type variable *)
  | sarr (s t : sty). (* function type *)

Inductive tm : Type :=
  | var (n : nat)           (* term variable *)
  | app (M N : tm)          (* application *)
  | lam (t : sty) (M : tm). (* type-annotated abstraction *)
```

In congruence with Section 3, the annotation `t` in the abstraction constructor `lam` is the simple type assigned to the whole term (not just the bound variable).

The proposition `stlc Gamma M t` mechanizes that the term `M` is assigned the simple type `t` in the simple type environment `Gamma`. $I\gamma K'$-normalization (Theorem 15) of simply typed terms is mechanized in `stlc_nf.v` as follows.

```
Theorem stlc_nf M Gamma t : stlc Gamma M t -> exists N, steps M N /\ nf N.
```

In the above, `steps M N` mechanizes $\mathtt{M} \to^*_{I\gamma K'} \mathtt{N}$, and `nf N` mechanizes that the term `N` is β-normal.

[2] `https://github.com/tudo-seal/uniform-intersection`

Non-idempotent intersection types are mechanized in `nitlc.v` as `nity`.

```
Inductive nity : Type :=
  | niatom (x : nat)                    (* type variable *)
  | niarr (u : list nity) (A : nity).  (* function type *)
```

The proposition `nitlc Gamma M A` mechanizes that the term `M` is assigned the uniform intersection type `A` in the uniform intersection type environment `Gamma`. Simple type typability of terms which can be assigned a uniform intersection type (Lemma 23) is mechanized in `nitlc_typ.v` as follows.

```
Theorem nitlc_stlc Gamma0 M Gamma A :
  nitlc Gamma M A ->
  env_ssim Gamma0 Gamma ->
  allfv (fun x => nth_error Gamma0 x <> None) M ->
  exists t, stlc Gamma0 M t /\ ssim t A.
```

In the above, the pointwise collapse of the uniform intersection type environment `Gamma` to the simple type environment `Gamma0` is mechanized by `env_ssim Gamma0 Gamma` together with `allfv (fun x => nth_error Gamma0 x <> None) M`. The proposition `ssim t A` mechanizes that the non-idempotent type `A` is uniform and collapses to the existentially quantified simple type `t` (cf. Definition 19).

Finally, uniform typability of simply typed terms (Theorem 32) is mechanized in `nitlc_typ.v` as follows.

```
Theorem nitlc_type_inference M Gamma0 t : stlc Gamma0 M t ->
  exists Gamma A,
    nitlc Gamma M A /\
    Forall2 (fun s u => u <> [] /\ Forall (ssim s) u) Gamma0 Gamma /\
    ssim t A.
```

In the above, if a term `M` is assigned a simple type `t` in the simple type environment `Gamma0`, then there exists a uniform intersection type environment `Gamma` and a uniform intersection type `A` such that the following conditions hold:

- The term `M` is assigned the type `A` in the environment `Gamma`.
- Each multiset `u` in `Gamma` is nonempty and collapses to the corresponding simple type `s` in `Gamma0`.
- The type `A` collapses to the simple type `t`.

The proof structure of the above `Theorem nitlc_type_inference` relies on the mechanization of typability preserving expansion, namely `Theorem stepI_expansion` (Lemma 27), `Theorem stepG_expansion` (Lemma 30), and `Theorem stepK_expansion` (Lemma 31).

There are three interdependent aspects of the proof of `Theorem nitlc_type_inference` which highlight the utility of the Coq proof assistant.

- The definition of the K'-reduction (Definition 1) allows for an expansion lemma.
- The inductive hypothesis for the expansion lemma (cf. Lemma 31) is chosen carefully.
- The particular inductive proof involves extensive, nested case analyses for the chosen definition and inductive hypothesis.

In all three aspects the development of the proof was guided by the proof assistant: the technical details listed by the tool motivated the particular definition of the K'-reduction. In fact, the proof was developed via interaction with the mechanized statement prior to being transcribed into a traditional written format.

7 Conclusion

By providing an alternative proof that all simply typable terms can be assigned a quantitative type by system $\mathcal{U}$ (Theorem 32), we are able to easily infer strong normalization of STLC (Theorem 34). The presented typability proof, fully formalized in Coq, is constructive: this means that an actual type inference algorithm for system $\mathcal{U}$ can be extracted from it. Such an algorithm is conceptually dual to the one proposed by [23]; indeed, the technique presented in the work at hand focuses on term expansion rather than on term reduction.

Inductive proofs of subject expansion in both systems $\mathcal{U}$ and $\mathcal{I}$ (most notably Lemma 31 and Lemma 38) are achieved by means of the introduced $I\gamma K'$-reduction, for which we show that simply typed terms are normalizing (Theorem 15). In addition, our perspective on SN leads to the discovery of an interesting family of perpetual reduction strategies (Corollary 40).

The present work also highlights the role of Coq as a proof *assistant*: its contribution was crucial in the design of the $I\gamma K'$-reduction and the mechanical verification of technical details of the aforementioned results.

References

1 Pablo Barenbaum and Cristian Sottile. Two decreasing measures for simply typed λ-terms. In Marco Gaboardi and Femke van Raamsdonk, editors, *8th International Conference on Formal Structures for Computation and Deduction, FSCD 2023, July 3-6, 2023, Rome, Italy*, volume 260 of *LIPIcs*, pages 11:1–11:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.FSCD.2023.11.

2 Hendrik Pieter Barendregt, Wil Dekkers, and Richard Statman. *Lambda Calculus with Types*. Perspectives in logic. Cambridge University Press, 2013. URL: http://www.cambridge.org/de/academic/subjects/mathematics/logic-categories-and-sets/lambda-calculus-types.

3 Antonio Bucciarelli, Delia Kesner, and Simona Ronchi Della Rocca. Inhabitation for non-idempotent intersection types. *Log. Methods Comput. Sci.*, 14(3), 2018. doi:10.23638/LMCS-14(3:7)2018.

4 Antonio Bucciarelli, Delia Kesner, and Daniel Ventura. Non-idempotent intersection types for the lambda-calculus. *Log. J. IGPL*, 25(4):431–464, 2017. doi:10.1093/JIGPAL/JZX018.

5 Mario Coppo and Mariangiola Dezani-Ciancaglini. A new type assignment for λ-terms. *Arch. Math. Log.*, 19(1):139–156, 1978. doi:10.1007/BF02011875.

6 Mario Coppo and Mariangiola Dezani-Ciancaglini. An extension of the basic functionality theory for the lambda-calculus. *Notre Dame Journal of Formal Logic*, 21(4):685–693, 1980. doi:10.1305/ndjfl/1093883253.

7 Mario Coppo, Mariangiola Dezani-Ciancaglini, and Betti Venneri. Functional characters of solvable terms. *Mathematical Logic Quarterly*, 27(2-6):45–58, 1981. doi:10.1002/malq.19810270205.

8 René David. Normalization without reducibility. *Ann. Pure Appl. Log.*, 107(1-3):121–130, 2001. doi:10.1016/S0168-0072(00)00030-0.

9 Nicolaas Govert De Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. In *Indagationes Mathematicae (Proceedings)*, volume 75, pages 381–392. North-Holland, 1972.

10 Daniel de Carvalho. Execution time of λ-terms via denotational semantics and intersection types. *Math. Struct. Comput. Sci.*, 28(7):1169–1203, 2018. doi:10.1017/S0960129516000396.

11 Philippe de Groote. The conservation theorem revisited. In Marc Bezem and Jan Friso Groote, editors, *Typed Lambda Calculi and Applications, International Conference on Typed Lambda Calculi and Applications, TLCA '93, Utrecht, The Netherlands, March 16-18, 1993, Proceedings*, volume 664 of *Lecture Notes in Computer Science*, pages 163–178. Springer, 1993. doi:10.1007/BFB0037105.

12 Nachum Dershowitz and Zohar Manna. Proving termination with multiset orderings. *Commun. ACM*, 22(8):465–476, 1979. doi:10.1145/359138.359142.

13 Jean H. Gallier. *On Girard's "Candidats de Reductibilité"*. University of Pennsylvania, 1989. URL: https://api.semanticscholar.org/CorpusID:14688391.

14 Philippa Gardner. Discovering needed reductions using type theory. In Masami Hagiya and John C. Mitchell, editors, *Theoretical Aspects of Computer Software, International Conference TACS '94, Sendai, Japan, April 19-22, 1994, Proceedings*, volume 789 of *Lecture Notes in Computer Science*, pages 555–574. Springer, 1994. doi:10.1007/3-540-57887-0_115.

15 Jean-Yves 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 J.E. Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, volume 63 of *Studies in Logic and the Foundations of Mathematics*, pages 63–92. Elsevier, 1971. doi:10.1016/S0049-237X(08)70843-7.

16 Inge Li Gørtz, Signe Reuss, and Morten Heine Sørensen. Strong normalization from weak normalization by translation into the lambda-I-calculus. *High. Order Symb. Comput.*, 16(3):253–285, 2003. doi:10.1023/A:1025693307470.

17 Assaf J. Kfoury and Joe B. Wells. Addendum to "New notions of reduction and non-semantic proofs of strong beta-normalization in typed lambda calculi", 1995. URL: https://open.bu.edu/handle/2144/1568.

18 Assaf J. Kfoury and Joe B. Wells. New notions of reduction and non-semantic proofs of beta-strong normalization in typed lambda-calculi. In *Proceedings, 10th Annual IEEE Symposium on Logic in Computer Science, San Diego, California, USA, June 26-29, 1995*, pages 311–321. IEEE Computer Society, 1995. doi:10.1109/LICS.1995.523266.

19 Jan Willem Klop. *Combinatory reduction systems*. PhD thesis, Univ. Utrecht, 1980.

20 Robert Pieter Nederpelt Lazarom. *Strong normalization in a typed lambda calculus with lambda structured types*. PhD thesis, TU Eindhoven, 1973.

21 Jean-Jacques Lévy. *Réductions correctes et optimales dans le lambda-calcul*. PhD thesis, Université de Paris 7, 1978.

22 Peter Møller Neergaard. Theoretical pearls: A bargain for intersection types: a simple strong normalization proof. *J. Funct. Program.*, 15(5):669–677, 2005. doi:10.1017/S0956796805005587.

23 Daniele Pautasso and Simona Ronchi Della Rocca. A quantitative version of simple types. In Marco Gaboardi and Femke van Raamsdonk, editors, *8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023)*, volume 260 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 29:1–29:21, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.FSCD.2023.29.

24 Garrel Pottinger. A type assignment for the strongly normalizable λ-terms. *To HB Curry: essays on combinatory logic, lambda calculus and formalism*, pages 561–577, 1980.

25 Helmut Schwichtenberg. An upper bound for reduction sequences in the typed λ-calculus. *Arch. Math. Log.*, 30(5-6):405–408, 1991. doi:10.1007/BF01621476.

26 Morten Heine Sørensen. Strong normalization from weak normalization in typed lambda-calculi. *Inf. Comput.*, 133(1):35–71, 1997. doi:10.1006/INCO.1996.2622.

27 Kathrin Stark, Steven Schäfer, and Jonas Kaiser. Autosubst 2: reasoning with multi-sorted de Bruijn terms and vector substitutions. In Assia Mahboubi and Magnus O. Myreen, editors, *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019, Cascais, Portugal, January 14-15, 2019*, pages 166–180. ACM, 2019. doi:10.1145/3293880.3294101.

28 William W. Tait. Intensional interpretations of functionals of finite type I. *Journal of Symbolic Logic*, 32(2):198–212, 1967. doi:10.2307/2271658.

29 William W. Tait. A realizability interpretation of the theory of species. In Rohit Parikh, editor, *Logic Colloquium*, pages 240–251, Berlin, Heidelberg, 1975. Springer Berlin Heidelberg.

30 The Coq Development Team. The Coq proof assistant, July 2023. doi:10.5281/zenodo.8161141.

31 Femke van Raamsdonk, Paula Severi, Morten Heine Sørensen, and Hongwei Xi. Perpetual reductions in lambda-calculus. *Inf. Comput.*, 149(2):173–225, 1999. doi:10.1006/INCO.1998.2750.

32 Hongwei Xi. Weak and strong beta normalisations in typed lambda-calculi. In Philippe de Groote, editor, *Typed Lambda Calculi and Applications, Third International Conference on Typed Lambda Calculi and Applications, TLCA '97, Nancy, France, April 2-4, 1997, Proceedings*, volume 1210 of *Lecture Notes in Computer Science*, pages 390–404. Springer, 1997. doi:10.1007/3-540-62688-3_48.

Laplace Distributors and Laplace Transformations for Differential Categories

Marie Kerjean
CNRS, Université Sorbonne Paris Nord, France

Jean-Simon Pacaud Lemay
School of Mathematical and Physical Sciences, Macquarie University, Sydney, Australia

Abstract

In a differential category and in Differential Linear Logic, the exponential conjunction ! admits structural maps, characterizing quantitative operations and symmetric co-structural maps, characterizing differentiation. In this paper, we introduce the notion of a Laplace distributor, which is an extra structural map which distributes the linear negation operation $(\_)^*$ over ! and transforms the co-structural rules into the structural rules. Laplace distributors are directly inspired by the well-known Laplace transform, which is all-important in numerical analysis. In the star-autonomous setting, a Laplace distributor induces a natural transformation from ! to the exponential disjunction ?, which we then call a Laplace transformation. According to its semantics, we show that Laplace distributors correspond precisely to the notion of a generalized exponential function e^x on the monoidal unit. We also show that many well-known and important examples have a Laplace distributor/transformation, including (weighted) relations, finiteness spaces, Köthe spaces, and convenient vector spaces.

2012 ACM Subject Classification Theory of computation → Categorical semantics; Theory of computation → Denotational semantics; Theory of computation → Linear logic

Keywords and phrases Differential Categories, Differential Linear Logic, Laplace Distributor, Laplace Transformation, Exponential Function

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.9

Funding *Marie Kerjean*: Part of the ANR DIFFERENCE # ANR-20-CE48-0002 and the ANR NuSCAP # ANR-20-CE48-0014.
Jean-Simon Pacaud Lemay: ARC DECRA (DE230100303) & AFOSR (FA9550-24-1-0008)

Acknowledgements We are grateful to Yoann Dabrowski for enriching discussions on Laplace and Fourier transformations.

1 Introduction

Differential Linear Logic (DiLL) [11], introduced by Ehrhard and Regnier [12], introduces the concept of differentiation in Linear Logic (LL), as introduced by Girard [15], by symmetrizing three out of the four rules for the aptly called exponential connective !. So LL features four exponential structural rules which dictate the use of $!A$; they are: the weakening rule w, the contraction rule c, the dereliction rule d, and the promotion rule P[1].

$$\frac{\Gamma \vdash \Delta}{\Gamma, !A \vdash \Delta}\,\mathsf{w} \qquad \frac{\Gamma, !A, !A \vdash \Delta}{\Gamma, !A \vdash \Delta}\,\mathsf{c} \qquad \frac{\Gamma, A \vdash \Delta}{\Gamma, !A \vdash \Delta}\,\mathsf{d} \qquad \frac{!\Gamma \vdash A}{!\Gamma \vdash !A}\,\mathsf{P}$$

It is worth mentioning that the promotion rule can be equivalently replaced by two rules: the functorial promotion rule $!_{\mathsf{f}}$ and the digging rule p.

$$\frac{\Gamma \vdash A}{!\Gamma \vdash !A}\,!_{\mathsf{f}} \qquad \frac{\Gamma, !!A \vdash \Delta}{\Gamma, !A \vdash \Delta}\,\mathsf{p}$$

[1] These rules are presented with bilateral sequent for simplicity, but they could also be made monolateral by using the exponential disjunction ?, which is the dual of !.

DiLL then adds the co-structural rules which are the co-weakening rule $\overline{\mathsf{w}}$, the co-contraction rule $\overline{\mathsf{c}}$, and the co-dereliction rule $\overline{\mathsf{d}}$.

$$\frac{\vdash}{\vdash !A}\,\overline{\mathsf{w}}\qquad \frac{\vdash \Gamma, !A \qquad \vdash \Delta, !A}{\vdash \Gamma, \Delta, !A}\,\overline{\mathsf{c}}\qquad \frac{\vdash \Gamma, A}{\vdash \Gamma, !A}\,\overline{\mathsf{d}}$$

The co-dereliction rule $\overline{\mathsf{d}}$ expresses differentiation, while the co-contraction rule $\overline{\mathsf{c}}$ and the co-weakening rule $\overline{\mathsf{w}}$ are necessary for cut-elimination purposes. This beautifully results in a symmetry between the structural rules and co-structural rules, that has however never been properly explained. In this paper, we explain this symmetry using the *Laplace transform*.

1.1 Differentiation on proofs

Before diving into the Laplace transform and its interpretation in categorical models of DiLL, let us give more intuitions on the co-structural rules of DiLL. The core intuition of LL is that a proof of a sequent $A \vdash B$ will be a *linear* proof, making use of A exactly once and not allowing contraction nor weakening on A. This is opposed to a proof of $!A \vdash B$, which can make a non-linear usage of A by using contraction or weakening. The basic rule of LL is that you can forget about linearity. Hence, the dereliction rule d transforms a linear proof into a non-linear proof, which intuitively is done so by just forgetting about the linearity property. DiLL takes the reverse path by introducing a co-dereliction rule $\overline{\mathsf{d}}$, which, after a cut, allows the transformation of a non-linear proof $!A \vdash B$ into a linear proof $A \vdash B$. From a semantical point of view, linearizing a non-linear function (which interprets a proof) is done so via differentiation. This analogy is made precise by introducing new cut-elimination rules between $\overline{\mathsf{d}}$ and structural rules. The cut-elimination between d and $\overline{\mathsf{d}}$ results in a cut between their premises, and this represents the fact that differentiating at 0 a linear function returns the same linear function. The cut-elimination between promotion p and $\overline{\mathsf{d}}$ is more intricate and uses $\overline{\mathsf{c}}$ and $\overline{\mathsf{w}}$: it represents the chain rule, which is the formula expressing how to differentiate a composition of functions.

Rules of DiLL can also be understood through the notions of functions and *distributions*. Naively, distributions are linear scalar maps which are computed on smooth functions. Let us for now suggestively denote $\mathcal{C}^\infty(A,B) := \mathcal{L}(!A, B)$ the set of smooth maps from A to B, and $A \multimap B := \mathcal{L}(A,B)$ the set of linear maps from A to B. Now, in Classical DiLL, elements of $!A$ can be interpreted as distributions, so we may suggestively write $!A \subseteq \mathcal{C}^\infty(A, I) \multimap I$. In most models, I is often interpreted as the field of real or complex numbers. Now, for each element x of A, the dereliction rule gives us the Dirac distribution at x, which is the distribution $\delta_x \in !A$ which evaluates a smooth function at x, so $\delta_x(f) = f(x)$. For finite-dimensional vector spaces, or in the model of convenient vector spaces [1] (which we discuss in Ex 14), it is sufficient to define what a non-linear map does on Dirac distributions. So, on Dirac distributions, the structural maps, which correspond to the structural rules of LL and the co-structural rules of DiLL, are given as follows:

$$\begin{aligned} &\mathsf{p}_A(\delta_x) = \delta_{\delta_x} \qquad \mathsf{d}_A(\delta_x) = x \qquad \mathsf{c}_A(\delta_x) = \delta_x \otimes \delta_x \qquad \mathsf{w}_A(\delta_x) = 1 \\ &\overline{\mathsf{d}}_A(x) = \mathsf{D}_0(\_)(x) \qquad \overline{\mathsf{c}}_A(\delta_x \otimes \delta_y) = \delta_{x+y} \qquad \overline{\mathsf{w}}_A(1) = \delta_0 \end{aligned} \tag{1}$$

where for the co-dereliction $\overline{\mathsf{d}}$, the D is the differential operator, that is, for a smooth function f, $\mathsf{D}_x(f)(y)$ is the derivative of f at point x along the vector y. We highlight that on the whole space $!A$, the co-contraction $\overline{\mathsf{c}} : !A \otimes !A \to !A$ is interpreted as the *convolution of distributions*:

$$\overline{\mathsf{c}}_A(\phi \otimes \psi) = \phi * \psi := f \mapsto \phi(x \mapsto \psi(y \mapsto f(x+y))).$$

Moreover, the structural rules of LL can also be naturally expressed on functions. Indeed, in Classical DiLL, we have an involutive duality $*$ where A^* is the linear dual of A, that is, $A^* = A \multimap I$. Using the linear dual, one also introduces the connector $?A = (!A^*)^*$, which is interpreted as a space of smooth functions, $?A \subseteq \mathcal{C}^\infty(A^*, I)$. We also get the multiplicative disjunction $A \parr B = (A^* \otimes B^*)^*$, which we may think of as a completed tensor product. Then the contraction $\mathsf{c}^?_A : ?A \parr ?A \to ?A$ is interpreted by the pointwise multiplication of scalar functions, the weakening $\mathsf{w}^?_A : \mathbb{K} \to ?A$ maps scalars r to constant functions $cst_r : x \mapsto r$, and the dereliction $\mathsf{d}^?_A : A \to ?A$ maps elements of A to their evalutation at a point x:

$$\mathsf{c}^?_A\,(f \otimes g) = f \cdot g \qquad \mathsf{w}^?_A(r) = \mathrm{cst}_r \qquad \mathsf{d}^?_A(x) = (\ell \in A^* \mapsto \ell(x)) \tag{2}$$

All these intuitions can be made formal in specific models of DiLL; see Section 6.

1.2 A higher-order Laplace transform

The categorification of functional analysis and differential geometry entertains close links with the semantics of the sequent calculus for LL and DiLL. Differential categories were introduced by Blute, Cocket, and Seely [3], and originated from the semantics of DiLL. Since their introduction, differential categories now have a rich mathematical literature and have been quite successful in categorifying various important concepts from differential calculus and differential geometry, as well as various other aspects of differentiation throughout mathematics and computer science. This paper follows this line of research. Following the categorification of the exponential functions in a differential category by the second named author in [20], and the completion of DiLL by the addition of a co-digging rule by the authors in [18], here we give a categorical interpretation of the Laplace transform and study its properties. We explain why it is the reason behind the symmetry in DiLL rules, which we exploit categorically.

The Laplace transform is a central component of calculus and engineering, as it changes differential equations into polynomial equations. As such, the Laplace transform is a very useful tool for solving differential equations. In its first-order version, the Laplace transform takes a function $f : \mathbb{R} \to \mathbb{R}$ to a function $\mathcal{L}(f) : \mathbb{C} \to \mathbb{R}$, defined as:

$$\mathcal{L}(f)(s) = \int_0^\infty f(t)e^{-st}\,dt$$

However, this first-order version does not necessarily fit well with the categorical semantics of DiLL. Instead of using integration to make functions act on functions, one can use distributions by following the general idea of interpreting distributions as generalized functions. Consider a distribution ϕ with compact support, that is, $\phi \in \mathcal{C}^\infty(\mathbb{R}, \mathbb{R})'$ is a linear form on the space of smooth functions (where $F' := \mathcal{L}(F, \mathbb{R})$ is the space of linear scalar functions on a vector space F). Then we may write:

$$\mathcal{L}(\phi)(s) = \phi(t \mapsto e^{-st})$$

So for a higher-order distribution $\phi \in \mathcal{C}^\infty(E, \mathbb{R})'$, where E stands for a possibly infinite-dimensional vector space, we get:

$$\mathcal{L} : \begin{cases} \mathcal{C}^\infty(E, \mathbb{R})' \longrightarrow \mathcal{C}^\infty(E', \mathbb{R}) \\ \qquad\qquad \phi \longmapsto \Big(x^* \mapsto \Big(\phi\Big(t \mapsto e^{x^*(t)}\Big)\Big)\Big) \end{cases} \tag{3}$$

Following the intuitions developed above, this gives us a new understanding of the Laplace transformation in terms of connectives of LL, resulting in a natural transformation of type $\mathcal{L}_A : !A \to ?A$. This idea was only recently noticed in the context of DiLL, thanks to the higher-order presentation of the Laplace transform in a specific polarized model of DiLL discussed in [18, Prop V.8].

1.3 Laplace transformation from co-structural to structural rules

Since we have a categorical understanding of higher-order distribution theory, we, therefore, have all the ingredients in hand to axiomatize $\mathcal{L}$ categorically in a suitable differential category. Differential categories are reviewed in Section 2. For now, we simply state that we categorify the Laplace transform as a natural transformation of type $\mathcal{L} : !A \to ?A$, which we call a **Laplace transformation** (Def 8). Semantically, the axioms say that $\mathcal{L}$ transforms the interpretation of co-structural rules into the interpretation of structural rules:

$$\mathcal{L} : !A \mapsto ?A; \overline{\mathsf{w}} \mapsto \mathsf{w}; \overline{\mathsf{c}} \mapsto \mathsf{c}; \overline{\mathsf{d}} \mapsto \mathsf{d}.$$

These are all analogues of very well-known facts in calculus. For example, the Laplace transform converts convolution into multiplication, which is recaptured by the fact that our Laplace transformation $\mathcal{L}$ turns $\overline{\mathsf{c}}$ into $\mathsf{c}^?$. It may be useful to redo these well-known computations, which will help clearly show how $\mathcal{L}$, as given in (3), computes on co-structural morphisms, as given in (1). Here x^* is an element of $A^* = \mathcal{L}(A, \mathbb{R})$, ans as such acts on elements t of A.

$$\mathcal{L}(\overline{\mathsf{w}}_A(r)) = x^* \mapsto r \cdot \delta_0(t \mapsto e^{x^*(t)}) = x^* \mapsto r \cdot e^0 = x^* \mapsto r = \mathsf{w}^?_A(r)$$

$$\mathcal{L}(\overline{\mathsf{d}}_A(y)) = x^* \mapsto D_0(t \mapsto e^{x^*(t)})(y) = x^* \mapsto x^*(y) = \mathsf{d}^?_A(y)$$

$$\begin{aligned}
\mathcal{L}(\overline{\mathsf{c}}_A(\phi \otimes \psi)) &= x^* \mapsto (\phi * \psi)(t \mapsto e^{x^*(t)}) = x^* \mapsto \phi\left(s \mapsto \psi(t \mapsto e^{x^*(t+s)})\right) \\
&= x^* \mapsto \phi\left(s \mapsto \psi(t \mapsto e^{x^*(t)+x^*(s))})\right) && (x^* \text{ is linear}) \\
&= x^* \mapsto \phi\left(s \mapsto \psi(t \mapsto e^{x^*(t)}e^{x^*(s)})\right) \\
&= x^* \mapsto \phi(s \mapsto e^{x^*(s)} \cdot \psi(t \mapsto e^{x^*(t)})) && (\psi \text{ is linear}) \\
&= x^* \mapsto \phi(s \mapsto e^{x^*(s)}) \cdot \psi(t \mapsto e^{x^*(t)}) && (\phi \text{ is linear}) \\
&= \mathsf{c}^?_{A^*}(\mathcal{L}(\phi) \otimes \mathcal{L}(\psi))
\end{aligned}$$

Observe how all these equations are intrinsically linked with the basic properties of the exponential function e^x. We will make this precise in Section 4. Indeed, generalizations of the exponential function in a differential category were defined by the second named author in [20], and are axiomatized by analogues of three fundamental properties of the exponential function: that $e^{x+y} = e^x e^y$ and $e^0 = 1$, and also that e^x is its own derivative. We will explain how the notion of a Laplace transformation is fundamentally linked to that of a generalized exponential function on the monoidal unit I (Def 3).

Moreover, since we are in the monoidal closed, we may uncurry the Laplace transformation to get an extranatural transformation $\partial_A : !A^* \otimes !A \to I$, which we call a **Laplace evaluator** (Def 2), or take the dual to get a natural transformation $\ell_A : !A^* \to (!A)^*$, which we call a **Laplace distributor** (Def 1).

1.4 Content and Outline

This paper starts in Section 2 with a review of differential categories, the categorical semantics of DiLL, in which we will categorify $\mathcal{L}$. In Section 3, we introduce the concept of a Laplace distributor, which we axiomatize as a natural transformation operating in differential linear closed categories which transforms co-structural rules into structural rules. In Section 4, we show that the presence of a Laplace distributor in a differential linear closed category is equivalent to the presence of a generalized exponential function $\mathsf{e} : !I \multimap I$ on the monoidal unit. Laplace distributors in the context of isomix star-autonomous categories are studied in Section 5, where we show that we obtain our desired Laplace transformation $\mathcal{L}$ in a differential linear isomix category. In Section 6, we give examples of Laplace distributors/evaluators/transformations in well-known differential categories. We then conclude in Section 7 with a discussion of future work.

2 Background: Differential Categories

In this section, we quickly review differential categories, mostly to set terminology and notation. We will follow the same terminology and notation used in [18]. For a more in-depth introduction to the basics of monoidal categories and the overall categorical semantics of linear logic, we refer the reader to see [22], and for an in-depth introduction to differential categories and examples, we refer them to see [2, 11].

The underlying categorical structure of a differential category is that of an *additive symmetric monoidal category*. Recall that a **symmetric monoidal category** [22, Sec 4.4] interprets the multiplicative fragment of LL. So for an arbitrary symmetric monoidal category, we denote the underlying category as $\mathcal{C}$, the monoidal product as $\otimes$, the monoidal unit as I, the natural associativity isomorphism as $\alpha_{A,B,C} : A \otimes (B \otimes C) \to (A \otimes B) \otimes C$, the natural right unital isomorphism as $\rho_A : A \otimes I \to A$, and the natural symmetry isomorphism as $\sigma_{A,B} : A \otimes B \to B \otimes A$. So then an **additive symmetric monoidal category** [2, Def 3] is a symmetric monoidal category $\mathcal{C}$ which is enriched over the category of commutative monoids, that is, each homset $\mathcal{C}(A, B)$ is a commutative monoid, with addition operation $+$ and zero $0 : A \to B$, and such that composition and the monoidal product $\otimes$ are compatible with the additive structure. This extra structure of additive enrichment for a differential category is necessary to express the famous Leibniz rule from differential calculus.

The categorical interpretation of the exponential fragment of LL is given by a *monoidal coalgebra modality*. So for a symmetric monoidal category $\mathcal{C}$, a **coalgebra modality** [2, Def 1] is a comonad $! : \mathcal{C} \to \mathcal{C}$ with comultiplication $\mathsf{p}_A : !A \to !!A$ called the **digging** and counit $\mathsf{d}_A : !A \to A$ called the **dereliction**, which comes equipped with two other natural transformations: $\mathsf{c}_A : !A \to !A \otimes !A$ called the **contraction** and $\mathsf{w}_A : !A \to I$ called the **weakening**, making each $!A$ a cocommutative comonoid and the digging a comonoid morphism. Then a **monoidal coalgbera modality** [2, Def 2] is a coalgebra modality $!$ which furthermore comes equipped with a natural transformation $\mu_{A,B} : !A \otimes !B \to !(A \otimes B)$ and a map $\mu_I : I \to !I$ which makes $!$ into a lax monoidal functor; p, d, c, and w into monoidal transformations; and c, and w into $!$-coalgebra morphisms.

For an additive symmetric monoidal category, a monoidal coalgebra modality can equivalently be described in terms of an **additive bialgebra modality** [2, Def 5]. So in particular, for a monoidal coalgebra modality $!$ on an additive symmetric monoidal category, we can build natural transformations $\overline{\mathsf{c}}_A : !A \otimes !A \to !A$ called the **co-contraction** and $\overline{\mathsf{w}}_A : I \to !A$ called the **co-weakening**, which in particular makes every $!A$ a commutative monoid, and in fact a bimonoid [2, Prop 1]. Then a **monoidal differential modality** is a monoidal coalgebra

modality (equiv. additive bialgebra modality) ! on an additive symmetric monoidal category which comes equipped with a natural transformation $\overline{\mathsf{d}}_A : A \to !A$ called a **co-dereliction** [2, Def 9], whose axioms are analogues of the fundamental rules of differential calculus such as the Leibniz rule and chain rule. Then a **differential linear category** [2, Sec 6] is an additive symmetric monoidal category equipped with a monoidal differential modality. One could also consider differential linear categories with finite products $\times$, which are called *differential storage categories*. In particular, in such a setting, we have the all-important *Seely isomorphisms* $!(A \times B) \cong !A \otimes !B$ [2, Def 10]. However, since products don't necessarily play a role in the story of this paper, we will not review them here and invite the reader to see [2, Sec 7] for details. Then a categorical model of (Classical) DiLL is a differential storage category that is also monoidal closed (resp. star-autonomous), which we discuss in Section 3 (resp. Section 5).

3 Laplace Distributor

In this section, we introduce the notion of a *Laplace distributor*, which is an extra structural natural transformation in a differential linear category that is also *closed*. To properly define a Laplace distributor, we will first have to set up some notation in the closed setting.

So for a **symmetric monoidal closed category** [22, Sec 4.7], we denote the internal homs by $A \multimap B$ and the evaluation map by $\epsilon_{A,B} : (A \multimap B) \otimes A \to B$. Explicitly, recall that closed means that for every map $f : C \otimes A \to B$, there exists a unique map $\lambda(f) : C \to A \multimap B$, called the **curry** of f, such that[2]:

$$(\lambda(f) \otimes 1_A); \epsilon_{A,B} = f \tag{4}$$

Now for every map $f : X \to A$ and $g : B \to Y$, we denote by $f \multimap g : A \multimap B \to X \multimap Y$ to be the unique map such that:

$$((f \multimap g) \otimes 1_X)\,; \epsilon_{X,Y} = (1_{A \multimap B} \otimes f); \epsilon_{A,B}; g \tag{5}$$

We note that for a monoidal coalgebra modality ! on a symmetric monoidal category, we have canonical maps $\xi_{A,B} : !(A \multimap B) \to !A \multimap !B$ defined as the unique map such that:

$$(\xi_{A,B} \otimes 1_{!A})\,; \epsilon_{!A,!B} = \mu_{A \multimap B,A}; !\epsilon_{A,B} \tag{6}$$

Then by a **differential linear closed category** we mean a differential linear category whose underlying symmetric monoidal category is closed.

Now, a Laplace distributor is a natural transformation that transforms the co-structural rules of the modality into its structural rule. This is expressed in terms of dual objects. In a symmetric monoidal closed category, the **dual** of an object A is the object $A^* := A \multimap I$. It is important to note that in an arbitrary symmetric monoidal closed category, the dual operation is not necessarily involutive, that is, A^{**} is not necessarily always equal/isomorphic to A. This will be a situation we discuss later in Section 5 below. We do however have a canonical isomorphism $\upsilon_I : I \to I^*$ which is defined as the unique map such that:

$$(\upsilon_I \otimes 1_I); \epsilon_{I,I} = \rho_I \qquad\qquad \upsilon_I^{-1} = \rho_{I^*}^{-1}; \epsilon_{I,I} \tag{7}$$

[2] In a category, we write identity maps as $1_A : A \to A$, and we write composition diagrammatically, that is, the composition of maps $f : A \to B$ and $g : B \to C$ is denoted $f; g : A \to C$.

as well another canonical isomorphism $\nu_{A,B} : (A \otimes B)^* \to A \multimap B^*$ defined as the unique map such that:

$$\begin{aligned} ((\nu_{A,B} \otimes 1_A) \otimes 1_B) \,;(\epsilon_{A,B^*} \otimes 1_B);\epsilon_{B,I} &= \alpha^{-1}_{(A\otimes B)^*,A,B};\epsilon_{A\otimes B,I} \\ \left(\nu^{-1}_{A,B} \otimes 1_{A\otimes B}\right);\epsilon_{A\otimes B,I} &= \alpha_{A\multimap B^*,A,B};(\epsilon_{A,B^*} \otimes 1_B);\epsilon_{B,I} \end{aligned} \tag{8}$$

Moreover, we also get canonical maps $\Theta_{A,B} : A^* \otimes B^* \to (A \otimes B)^*$ (which is not necessarily an isomorphism) defined as the unique map such that:

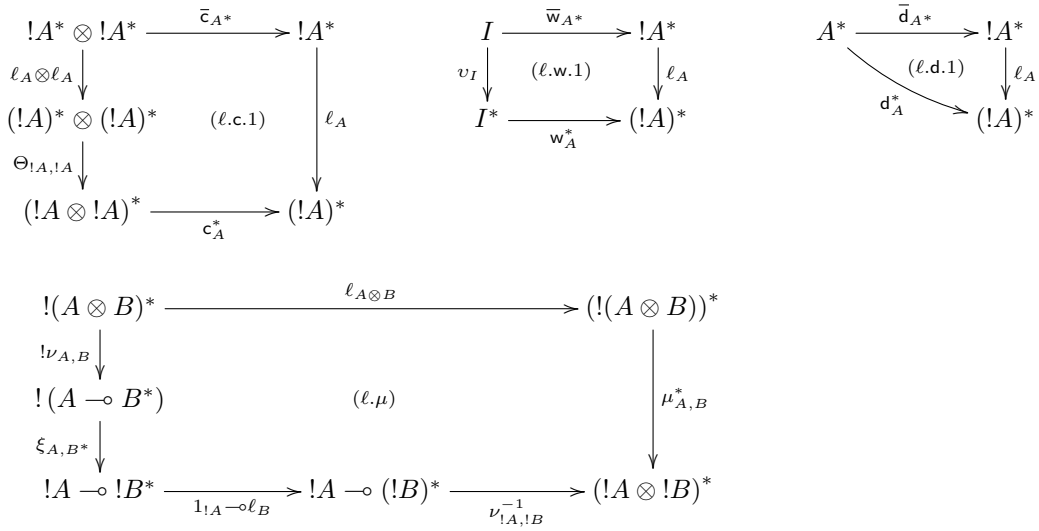
$$(\Theta_{A,B} \otimes 1_{A\otimes B}) \,;\epsilon_{A\otimes B,I} = \tau_{A^*,B^*,A,B};(\epsilon_{A,I} \otimes \epsilon_{B,I});\rho_I \tag{9}$$

where $\tau_{A,B,C,D} : (A\otimes B)\otimes(C\otimes D) \to (A\otimes C)\otimes(B\otimes D)$ is the canonical natural isomorphism which swaps the middle two terms. The dual operation also induces a contravariant functor, where in particular for every map $f : A \to B$, we also have a map of dual type $f^* : B^* \to A^*$ which is defined as the unique map such that:

$$(f^* \otimes 1_A);\epsilon_{A,I} = (1_{B^*} \otimes f);\epsilon_{B,I} \tag{10}$$

Then a Laplace distributor is a natural transformation which distributes $*$ over !, hence the name, and associates the co-structural maps to the dual of their mirror structural map.

▶ **Definition 1.** *For a differential linear closed category, a* ***Laplace distributor*** *is a natural transformation $\ell_A : !A^* \to (!A)^*$ such that the following diagrams commute:*



Examples of Laplace distributors can be found in Section 6. Let us provide some intuition for a Laplace distributor using our distribution analogy. First, for every linear functional $x^* : A \multimap I$, we have the Dirac distribution $\delta_{x^*} \in !A^*$. So the Laplace distributor produces a linear functional $\ell_A(\delta_{x^*}) : !A^* \multimap I$, which through the call-by-name translation of Linear Logic in Intuitionistic Logic corresponds to a smooth function $A \Rightarrow I$. Then for $z \in A$ and $x^*, y^* \in A$, the first three axioms of a Laplace distributor say that:

$$\ell_A(\delta_{x^*+y^*})(\delta_z) = \ell_A(\delta_{y^*})(\delta_z) \cdot \ell_A(\delta_{x^*})(\delta_z) \tag{ℓ.c.1}$$

$$\ell_A(\delta_0)(\delta_z) = 1 \tag{ℓ.w.1}$$

$$\ell_A(\mathsf{D}_0(\_)(x^*))(\delta_z) = x^*(z) \tag{ℓ.d.1}$$

Note the similarities with some of the basic identities the exponential function e^x satisfies. We will make this connection precise in Section 4 when we show how Laplace distributors correspond to a generalized version of the exponential function in a differential linear category. The last axiom ($\ell.\mu$) essentially tells us that every Laplace distributor can indeed be described as one of these generalized exponential functions. Somewhat surprisingly, we also get that the Laplace distributor also "co-transforms" the structural rules into their mirror co-structural rules, as we will see in Lemma 7 below.

Now the keen eyed-reader will note that in an arbitrary differential linear closed category, there are always two possible natural transformations of type $!A^* \to (!A)^*$ given by the following composites:

$$!A^* \xrightarrow{\mathsf{w}_{A^*}} I \xrightarrow{\upsilon_I} I^* \xrightarrow{\mathsf{w}_A^*} (!A)^* \qquad !A^* \xrightarrow{\mathsf{d}_{A^*}} A^* \xrightarrow{\mathsf{d}_A^*} (!A)^*$$

However, since by [2, Lemma 2] we have that $\overline{\mathsf{w}};\mathsf{d} = 0$, the first map won't satisfy ($\ell.\mathsf{d}.1$), while by [2, Def 9] we have that $\overline{\mathsf{d}};\mathsf{w} = 0$, so the second map won't satisfy ($\ell.\mathsf{w}.1$). So a Laplace evaluator does not always necessarily exists, and is indeed extra structure. To further justify this fact, in Ex 13 we give an example of a differential linear closed category which does *not* have a Laplace distributor.

Moreover, note that given the type of a Laplace distributor $\ell_A : !A^* \to (!A)^*$, we can uncurry it to get a map of type $!A^* \otimes !A \to I$, which we call the *Laplace evaluator*.

▶ **Definition 2.** *In a differential linear closed category with a Laplace distributor ℓ, the **Laplace evaluator** is the extranatural transformation $\eth_A : !A^* \otimes !A \to I$ defined as the composite:*

$$\eth_A :\; = \; !A^* \otimes !A \xrightarrow{\ell_A \otimes 1_{!A}} (!A)^* \otimes !A \xrightarrow{\epsilon_{!A,I}} I \tag{11}$$

By extranaturality, we mean that for all maps $f : A \to B$, the following equality holds:

$$(!f^* \otimes 1_A);\ell_A = (1_{!B^*} \otimes !f);\eth_B \tag{12}$$

Of course, since the currying operation is an isomorphism, we could have alternatively and equivalently written this story in terms of the Laplace evaluator and defined the Laplace distributor as its curry, $\ell_A = \lambda(\eth_A)$.

4 Exponential Map

In this section, we show that Laplace distributors correspond precisely to a generalized version of the exponential function e^x on the monoidal unit. The generalization of e^x in a differential category was introduced by the second named author in [20] and was called an !-*differential exponential map*. An !-differential exponential map can be defined for any commutative monoid in a differential category and is axiomatized by analogues of the fact that e^x is its own derivative and is a monoid morphism from addition to multiplication. Since the monoidal unit I in a symmetric monoidal category is canonically a monoid, we can consider an !-differential exponential map on I, which we call an I-*exponential map* for short.

▶ **Definition 3.** *In a differential linear category, an **I-exponential map** [20, Def 14] is a map $\mathsf{e} : !I \to I$ such that the following diagrams commute:*

$$\begin{array}{ccc} !I \otimes !I & \xrightarrow{\overline{\mathsf{c}}_I} & !I \\ {\scriptstyle \mathsf{e}\otimes\mathsf{e}}\downarrow & (\mathsf{e}.\overline{\mathsf{c}}) & \downarrow{\scriptstyle \mathsf{e}} \\ I \otimes I & \xrightarrow[\rho_I]{} & I \end{array} \qquad \begin{array}{ccc} I & \xrightarrow{\overline{\mathsf{w}}_I} & !I \\ & {\scriptstyle (\mathsf{e}.\overline{\mathsf{w}})}\searrow & \downarrow{\scriptstyle \mathsf{e}} \\ & & I \end{array} \qquad \begin{array}{ccc} I & \xrightarrow{\overline{\mathsf{d}}_I} & !I \\ & {\scriptstyle (\mathsf{e}.\overline{\mathsf{d}})}\searrow & \downarrow{\scriptstyle \mathsf{e}} \\ & & I \end{array}$$

Using our distribution analogy, let us explain why an I-exponential map is indeed the correct generalization of the exponential function $e^x : \mathbb{R} \to \mathbb{R}$. Note that by its type, an I-exponential map e is a smooth map from I to I, just like how e^x is a smooth function from $\mathbb{R}$ to $\mathbb{R}$. Then the axioms of an I-exponential map are:

$$\mathsf{e}(\delta_{x+y}) = \mathsf{e}(\delta_x)\mathsf{e}(\delta_y) \tag{e.$\overline{\mathsf{c}}$}$$
$$\mathsf{e}(\delta_0) = 1 \tag{e.$\overline{\mathsf{w}}$}$$
$$\mathsf{e}(\mathsf{D}_0(\_)(x)) = x \tag{e.$\overline{\mathsf{d}}$}$$

Now suggestively writing $\mathsf{e}(\delta_x) = e^x$, the three axioms of an I-exponential map give us precisely the well-known identities of the exponential function which are:

$$e^{x+y} = e^x e^y \qquad e^0 = 1 \qquad \mathsf{D}_0(e^x)(x) = x$$

Recall that in the previous section, we suggested that the axioms of a Laplace distributor also corresponded to these three e^x identities. Here, we make this precise by showing that there is a bijective correspondence between Laplace distributors and I-exponential maps.

Starting from a Laplace distributor, we get an I-exponential map by considering the Laplace distributor at I and the fact that $I \cong I^*$:

▶ **Proposition 4.** *In a differential linear closed category with a Laplace distributor ℓ, define the map $\mathsf{e}^\ell : !I \to I$ as the following composite:*

$$\mathsf{e}^\ell := \; !I \xrightarrow{!\upsilon_I} !I^* \xrightarrow{\ell_I} (!I)^* \xrightarrow{\mu_I^*} I^* \xrightarrow{\upsilon_I^{-1}} I \tag{13}$$

Proof. The key to this proof is using the part from the definition of a monoidal coalgebra modality [2, Def 1] which says that d, c, and w are compatible with μ_I since they are monoidal transformations. So, by definition, we have:

$$\mu_I; \mathsf{d}_I = 1_I \qquad \mu_I; \mathsf{w}_I = 1_I \qquad \mu_I; \mathsf{c}_I = \rho_I^{-1}; (\mu_I \otimes \mu_I) \tag{14}$$

More precisely, for the calculations in this proof, we will need the dualized versions of the above identities:

$$\mathsf{d}_I^*; \mu_I^* = 1_{I^*} \qquad \mathsf{w}_I^*; \mu_I^* = 1_{I^*} \qquad \mathsf{c}_I^*; \mu_I^* = (\mu_I \otimes \mu_I)^*; (\rho_I^{-1})^* \tag{15}$$

So we first compute (e.$\overline{\mathsf{w}}$):

$$\overline{\mathsf{w}}_I; \mathsf{e}^\ell \overset{(13)}{=} \overline{\mathsf{w}}_I; !\upsilon_I; \ell_I; \mu_I^*; \upsilon_I^{-1} \overset{\text{nat.}}{=} \overline{\mathsf{w}}_{I^*}; \ell_I; \mu_I^*; \upsilon_I^{-1} \overset{(\ell.\mathsf{w}.1)}{=} \upsilon_I; \mathsf{w}_I^*; \mu_I^*; \upsilon_I^{-1} \overset{(15)}{=} \upsilon_I; \upsilon_I^{-1} = 1_I$$

So $\overline{\mathsf{w}}_I; \mathsf{e}^\ell = 1_I$. Next we compute (e.$\overline{\mathsf{d}}$):

$$\overline{\mathsf{d}}_I; \mathsf{e}^\ell \overset{(13)}{=} \overline{\mathsf{d}}_I; !\upsilon_I; \ell_I; \mu_I^*; \upsilon_I^{-1} \overset{\text{nat.}}{=} \upsilon_I; \overline{\mathsf{d}}_{I^*}; \ell_I; \mu_I^*; \upsilon_I^{-1} \overset{(\ell.\mathsf{d}.1)}{=} \upsilon_I; \mathsf{d}_I^*; \mu_I^*; \upsilon_I^{-1} \overset{(15)}{=} \upsilon_I; \upsilon_I^{-1} = 1_I$$

So $\overline{\mathsf{d}}_I; \mathsf{e}^\ell = 1_I$. Lastly, using that Θ is natural and:

$$\Theta_{I,I}; (\rho_I^{-1})^*; \upsilon_I^{-1} = (\upsilon_I^{-1} \otimes \upsilon_I^{-1}); \rho_I \tag{16}$$

which we leave to the reader to check for themselves, we compute (e.$\overline{\mathsf{c}}$):

$$\overline{\mathsf{c}}_I; \mathsf{e}^\ell \overset{(13)}{=} \overline{\mathsf{c}}_I; !\upsilon_I; \ell_I; \mu_I^*; \upsilon_I^{-1} \overset{\text{nat.}}{=} (!\upsilon_I \otimes !\upsilon_I); \overline{\mathsf{c}}_{I^*}; \ell_I; \mu_I^*; \upsilon_I^{-1}$$

$$\overset{(\ell.\mathsf{c}.1)}{=} (!\upsilon_I \otimes !\upsilon_I); (\ell_I \otimes \ell_I); \Theta_{!I,!I}; \mathsf{c}_I^*; \mu_I^*; \upsilon_I^{-1}$$
$$\overset{(15)}{=} (!\upsilon_I \otimes !\upsilon_I); (\ell_I \otimes \ell_I); \Theta_{!I,!I}; (\mu_I \otimes \mu_I)^*; (\rho_I^{-1})^*; \upsilon_I^{-1}$$
$$\overset{\text{nat.}}{=} (!\upsilon_I \otimes !\upsilon_I); (\ell_I \otimes \ell_I); (\mu_I^* \otimes \mu_I^*); \Theta_{I,I}; (\rho_I^{-1})^*; \upsilon_I^{-1}$$
$$\overset{(16)}{=} (!\upsilon_I \otimes !\upsilon_I); (\ell_I \otimes \ell_I); (\mu_I^* \otimes \mu_I^*); (\upsilon_I^{-1} \otimes \upsilon_I^{-1}); \rho_I \overset{(13)}{=} (\mathsf{e}^\ell \otimes \mathsf{e}^\ell); \rho_I$$

So $\overline{\mathsf{c}}_I; \mathsf{e}^\ell = (\mathsf{e}^\ell \otimes \mathsf{e}^\ell); \rho_I$. Therefore, we conclude that e^ℓ is an I-exponential map. ◀

It is worthwhile to remark that we did not need $(\ell.\mu)$ to show that e^ℓ is an I-exponential map. This axiom will be used in Thm 6 when we prove the bijective correspondence. Before that, we must explain how from an I-exponential map we obtain a Laplace distributor, which is constructed using the evaluation map as well.

▶ **Proposition 5.** *In a differential linear closed category, if $\mathsf{e} : !I \to I$ is an I-exponential map then define the map $\ell^{\mathsf{e}}_A : !A^* \to (!A)^*$ as the curry of the following composite:*

$$!A^* \otimes !A \xrightarrow{\mu_{A^*,A}} !(A^* \otimes A) \xrightarrow{!\epsilon_{A,I}} !I \xrightarrow{\mathsf{e}} I \tag{17}$$

In other words, ℓ^{e}_A is the unique map such that the following equality holds:

$$(\ell^{\mathsf{e}}_A \otimes 1_{!A}); \epsilon_{!A,I} = \mu_{A^*,A}; !\epsilon_{A,I}; \mathsf{e} \tag{18}$$

Then ℓ^{e} is a Laplace distributor. Moreover, its induced Laplace evaluator $\mathscr{I}^{\mathsf{e}} : !A^* \otimes !A \to I$ is precisely the composite (17).

Proof. In this proof, for readability, we omit the subscripts. The key to this proof is that in a symmetric monoidal closed category, the evaluation map is *monic* in its first argument, that is, if $(f \otimes 1); \epsilon = (g \otimes 1); \epsilon$ then $f = g$.

We must first show naturality. So we compute that:

$$(!f^* \otimes 1); (\ell^{\mathsf{e}} \otimes 1); \epsilon \overset{(18)}{=} (!f^* \otimes 1); \mu_{A^*,A}; \epsilon; \mathsf{e} \overset{\text{nat.}}{=} \mu; !(f^* \otimes 1); !\epsilon_{A,I}; \mathsf{e}$$
$$\overset{(10)}{=} \mu; !(1 \otimes f); !\epsilon; \mathsf{e} \overset{\text{nat.}}{=} (1 \otimes !f); \mu; !\epsilon; \mathsf{e} \overset{(18)}{=} (1 \otimes !f); (\ell^{\mathsf{e}} \otimes 1); \epsilon$$
$$= (\ell^{\mathsf{e}} \otimes 1); (1 \otimes !f); \epsilon \overset{(10)}{=} (\ell^{\mathsf{e}} \otimes 1); ((!f)^* \otimes 1); \epsilon$$

Thus we get that $!f^*; \ell^{\mathsf{e}} = \ell^{\mathsf{e}}; (!f)^*$, and therefore ℓ^{e} is indeed a natural transformation.

To prove the first three axioms of a Laplace distributor, we will need the following compatibility relation between μ and the co-structural maps from [2, Prop 2 & Prop 5]:

$$(\overline{\mathsf{w}} \otimes 1)\,; \mu = \rho; \mathsf{w}; \overline{\mathsf{w}} \quad \left(\overline{\mathsf{d}} \otimes 1\right); \mu = (1 \otimes \mathsf{d}); \overline{\mathsf{d}} \quad (\overline{\mathsf{c}} \otimes 1)\,; \mu = (1 \otimes 1 \otimes \mathsf{c}); \tau; (\mu \otimes \mu); \overline{\mathsf{c}} \tag{19}$$

So for $(\ell.\mathsf{w}.1)$, we compute that:

$$(\overline{\mathsf{w}} \otimes 1); (\ell^{\mathsf{e}} \otimes 1); \epsilon \overset{(18)}{=} (\overline{\mathsf{w}} \otimes 1)\,; \mu; !\epsilon; \mathsf{e} \overset{(19)}{=} \rho; \mathsf{w}; \overline{\mathsf{w}}; !\epsilon; \mathsf{e} \overset{\text{nat.}}{=} \rho; \mathsf{w}; \overline{\mathsf{w}}; \mathsf{e} \overset{(\mathsf{e}.\overline{\mathsf{w}})}{=} \rho; \mathsf{w}_A$$
$$\overset{\text{nat.}}{=} (1 \otimes \mathsf{w}); \rho \overset{(7)}{=} (1 \otimes \mathsf{w}); (\upsilon \otimes 1); \epsilon = (\upsilon \otimes 1); (1 \otimes \mathsf{w}); \epsilon \overset{(10)}{=} (\upsilon \otimes 1); (\mathsf{w}^* \otimes 1); \epsilon$$

Thus we get that $\overline{\mathsf{w}}; \ell^{\mathsf{e}} = \upsilon; \mathsf{w}^*$. Then for $(\ell.\mathsf{d}.1)$, we compute that:

$$(\overline{\mathsf{d}} \otimes 1); (\ell^{\mathsf{e}} \otimes 1); \epsilon \overset{(18)}{=} \left(\overline{\mathsf{d}} \otimes 1\right); \mu; !\epsilon; \mathsf{e} \overset{(19)}{=} (1 \otimes \mathsf{d}); \overline{\mathsf{d}}; !\epsilon; \mathsf{e}$$
$$\overset{\text{nat.}}{=} (1 \otimes \mathsf{d}); \epsilon; \overline{\mathsf{d}}; \mathsf{e} \overset{(\mathsf{e}.\overline{\mathsf{d}})}{=} (1 \otimes \mathsf{d}); \epsilon \overset{(10)}{=} (\mathsf{d}^* \otimes 1); \epsilon$$

Thus we get that $\overline{\mathsf{d}}_{A^*}; \ell^{\mathsf{e}}_A = \mathsf{d}^*_A$. Now for $(\ell.\mathsf{c}.1)$, we compute that:

$$(\overline{\mathsf{c}} \otimes 1); (\ell^{\mathsf{e}} \otimes 1); \epsilon_{!A,I} \overset{(18)}{=} (\overline{\mathsf{c}} \otimes 1) ; \mu; !\epsilon; \mathsf{e} \overset{(19)}{=} (1 \otimes \mathsf{c}) ; \tau; (\mu \otimes \mu); \overline{\mathsf{c}}; !\epsilon; \mathsf{e}$$
$$\overset{\text{nat.}}{=} (1 \otimes \mathsf{c}) ; \tau; (\mu \otimes \mu); (!\epsilon \otimes !\epsilon); \overline{\mathsf{c}}; \mathsf{e} \overset{(\mathsf{e}.\overline{\mathsf{c}})}{=} (1 \otimes \mathsf{c}) ; \tau; (\mu \otimes \mu); (!\epsilon \otimes !\epsilon); (\mathsf{e} \otimes \mathsf{e})$$
$$\overset{(18)}{=} (1 \otimes \mathsf{c}) ; \tau; ((\ell^{\mathsf{e}} \otimes 1) \otimes (\ell^{\mathsf{e}} \otimes 1)) ; (\epsilon \otimes \epsilon) \overset{\text{nat.}}{=} (1 \otimes \mathsf{c}) ; ((\ell^{\mathsf{e}} \otimes \ell^{\mathsf{e}}) \otimes 1) ; \tau; (\epsilon \otimes \epsilon)$$
$$\overset{(9)}{=} ((\ell^{\mathsf{e}} \otimes \ell^{\mathsf{e}}) \otimes 1) ; (\Theta \otimes 1) ; (1 \otimes \mathsf{c}) ; \epsilon \overset{(10)}{=} ((\ell^{\mathsf{e}} \otimes \ell^{\mathsf{e}}) \otimes 1) ; (\Theta \otimes 1) ; (\mathsf{c}^* \otimes 1) ; \epsilon$$

Thus we get that $\overline{\mathsf{c}}_{A^*}; \ell^{\mathsf{e}}_A = (\ell^{\mathsf{e}}_A \otimes \ell^{\mathsf{e}}_A); \Theta_{!A,!A}; \mathsf{c}^*_A$. Lastly, for $(\ell.\mu)$, we will need the monoidal associativity axiom from the definition of a monoidal coalgebra modality [2, Def 2]:

$$\alpha; (\mu \otimes 1); \mu = (1 \otimes \mu); \mu; !\alpha \tag{20}$$

Then we compute that:

$$(!\nu \otimes 1); (\xi \otimes 1); ((1 \multimap \ell^{\mathsf{e}}) \otimes 1); \left(\nu^{-1} \otimes 1\right) ; \epsilon$$
$$\overset{(8)}{=} (!\nu \otimes 1); (\xi \otimes 1); ((1 \multimap \ell^{\mathsf{e}}) \otimes 1); \alpha; (\epsilon \otimes 1); \epsilon$$
$$\overset{\text{nat.}}{=} \alpha; ((!\nu \otimes 1) \otimes 1) ; ((\xi \otimes 1) \otimes 1) ; (((1 \multimap \ell^{\mathsf{e}}) \otimes 1) \otimes 1) ; (\epsilon \otimes 1); \epsilon$$
$$\overset{(5)}{=} \alpha; ((!\nu \otimes 1) \otimes 1) ; ((\xi \otimes 1) \otimes 1) ; (\epsilon \otimes 1) ; \ell^{\mathsf{e}}; \epsilon$$
$$\overset{(8)}{=} \alpha; ((!\nu \otimes 1) \otimes 1) ; (\mu \otimes 1) ; (!\epsilon \otimes 1) ; \ell^{\mathsf{e}}; \epsilon$$
$$\overset{\text{nat.}}{=} \alpha; (\mu \otimes 1) ; (!(\nu \otimes 1) \otimes 1) ; (!\epsilon \otimes 1) ; \ell^{\mathsf{e}}; \epsilon$$
$$\overset{(18)}{=} \alpha; (\mu \otimes 1) ; (!(\nu \otimes 1) \otimes 1) ; (!\epsilon \otimes 1) ; \mu; !\epsilon; \mathsf{e}$$
$$\overset{\text{nat.}}{=} \alpha; (\mu \otimes 1) ; \mu; ! ((\nu \otimes 1) \otimes 1)) ; !(\epsilon \otimes 1); !\epsilon; \mathsf{e}$$
$$\overset{(20)}{=} (1 \otimes \mu) ; \mu; !\alpha; ! ((\nu \otimes 1) \otimes 1)) ; !(\epsilon \otimes 1); !\epsilon; \mathsf{e} \overset{(8)}{=} (1 \otimes \mu) ; \mu; !\epsilon; \mathsf{e}$$
$$\overset{(18)}{=} (1 \otimes \mu) ; (\ell^{\mathsf{e}} \otimes 1) ; \epsilon = (\ell^{\mathsf{e}} \otimes 1) ; (1 \otimes \mu) ; \epsilon \overset{(10)}{=} (\ell^{\mathsf{e}} \otimes 1) ; (\mu^* \otimes 1) ; \epsilon$$

Thus we get that $!\nu_{A,B}; \xi_{A,B^*}; 1_{!A} \multimap \ell^{\mathsf{e}}_B; \nu^{-1}_{!A,!B} = \ell^{\mathsf{e}}_{A \otimes B}; \mu^*_{A,B}$. Therefore, we conclude that ℓ^{e} is a Laplace distributor. By definition, we then get that (17) is indeed the induced Laplace evaluator. ◀

The constructions from the above two propositions are inverses of each other, thus giving us our desired bijective correspondence.

▶ **Theorem 6.** *For a differential linear closed category, there is a bijective correspondence between Laplace distributors and I-exponential maps.*

Proof. To show that the constructions from Prop 4 and Prop 5 are invereses of each other, we must show that $\mathsf{e}^{\ell^{\mathsf{e}}} = \mathsf{e}$ and $\ell^{\mathsf{e}^{\ell}} = \ell$. For the former, we will need the monoidal unital axiom from the definition of a monoidal coalgebra modality [2, Def 2]:

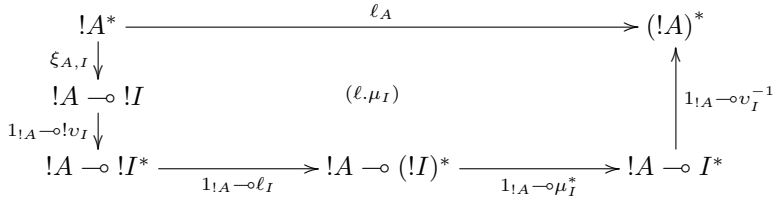
$$(1_{!A} \otimes \mu_I); \mu_{A,I} = \rho_{!A}; !(\rho^{-1}_A) \tag{21}$$

So we first compute that:

$$(\mathsf{e}^{\ell^{\mathsf{e}}} \otimes 1_I); (\upsilon_I \otimes 1); \epsilon_{I,I} \overset{(13)}{=} (!\upsilon_I \otimes 1_I); (\ell^{\mathsf{e}}_I \otimes 1_I); (\mu^*_I \otimes 1_I); (\upsilon^{-1}_I \otimes 1_I); (\upsilon_I \otimes 1); \epsilon_{I,I}$$
$$(!\upsilon_I \otimes 1_I); (\ell^{\mathsf{e}}_I \otimes 1_I); (\mu^*_I \otimes 1_I); \epsilon_{I,I} \overset{(10)}{=} (!\upsilon_I \otimes 1_I); (\ell^{\mathsf{e}}_I \otimes 1_I); (1_{(!I)^*} \otimes \mu_I); \epsilon_{!I,I}$$

$$= (!\upsilon_I \otimes 1_I); (1_{!I^*} \otimes \mu_I); (\ell^{\mathsf{e}}_I \otimes 1_{!I}); \epsilon_{!I,I} \overset{(18)}{=} (!\upsilon_I \otimes 1_I); (1_{!I^*} \otimes \mu_I); \mu_{I^*,I}; !\epsilon_{I^*,I}; \mathsf{e}$$
$$\overset{(21)}{=} (!\upsilon_I \otimes 1_I); \rho_{!I^*}; !(\rho^{-1}_{I^*}); !\epsilon_{I^*,I}; \mathsf{e} \overset{\text{nat.}}{=} \rho_{!I}; !\upsilon_I; !(\rho^{-1}_{I^*}); !\epsilon_{I^*,I}; \mathsf{e} \overset{(7)}{=} \rho_{!I}; !\upsilon_I; !\upsilon_I^{-1}; \mathsf{e} = \rho_{!I}; \mathsf{e}$$
$$\overset{\text{nat.}}{=} (\mathsf{e} \otimes 1_I); \rho_I \overset{(7)}{=} (\mathsf{e} \otimes 1_I); (\upsilon_I \otimes 1); \epsilon_{I,I}$$

So we get that $\mathsf{e}^{\ell^{\mathsf{e}}}; \upsilon_I = \mathsf{e}; \upsilon_I$, and since υ_I is an isomorphism, we get that $\mathsf{e}^{\ell^{\mathsf{e}}} = \mathsf{e}$. On the other hand, we leave it as an exercise for the reader to check that when taking $B = I$ in $(\ell.\mu)$ it follows that the following diagram commutes:



From this, we compute that:

$$(\ell^{\mathsf{e}^\ell}_A \otimes 1_{!A}); \epsilon_{!A,I} \overset{(18)}{=} \mu_{A^*,A}; !\epsilon_{A,I}; \mathsf{e}^\ell \overset{(13)}{=} \mu_{A^*,A}; !\epsilon_{A,I}; !\upsilon_I; \ell_I; \mu^*_I; \upsilon_I^{-1}$$
$$\overset{(6)}{=} (\xi_{A,I} \otimes 1_{!A}); \epsilon_{!A,!I}; !\upsilon_I; \ell_I; \mu^*_I; \upsilon_I^{-1}$$
$$\overset{(5)}{=} (\xi_{A,I} \otimes 1_{!A}); ((1_{!A} \multimap !\upsilon_I) \otimes 1_{!A}); ((1_{!A} \multimap \ell_I) \otimes 1_{!A}); ((1_{!A} \multimap !\upsilon_I) \otimes \mu^*_I); \left((1_{!A} \multimap \upsilon_I^{-1}) \otimes 1_{!A}\right); \epsilon_{!A,I}$$
$$\overset{(\ell.\mu_I)}{=} (\ell_A \otimes 1_{!A}); \epsilon_{!A,I}$$

So we get that $\ell^{\mathsf{e}^\ell}_A = \ell_A$. ◀

So now that we have proven that Laplace distributors do indeed correspond precisely to generalized versions of the exponential functions, let us revisit our distribution intuition for the axioms of a Laplace distributor. So suppose we have an I-exponential map, which recall we wrote as $\mathsf{e}(\delta_x) = e^x$. Then for every linear functional $x^* : A \multimap I$ and $z \in A$, the induced Laplace distributor is given as follows, which also corresponds to the axiom $(\ell.\mu)$:

$$\ell_A(\delta_{x^*})(\delta_z) = e^{x^*(z)} \tag{$\ell.\mu$}$$

Then the three other axioms of a Laplace distributor do indeed correspond to the three main identities of the exponential function:

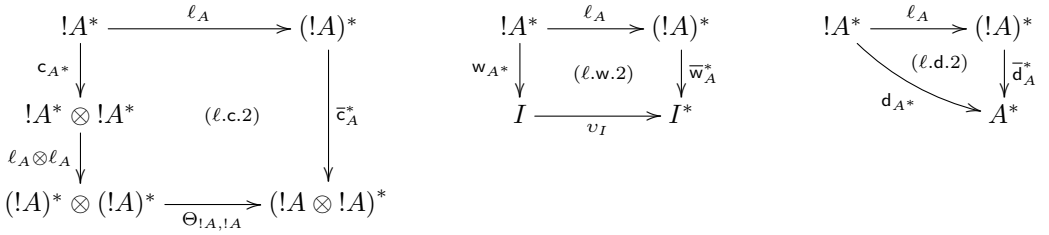
$$e^{x^*(z)+y^*(z)} = e^{x^*(z)} e^{y^*(z)} \tag{ℓ.c.1}$$
$$e^{0(z)} = 1 \tag{ℓ.w.1}$$
$$\mathsf{D}_0(e^x)(x^*(z)) = x^*(z) \tag{ℓ.d.1}$$

We can also describe the Laplace evaluator on Dirac distributions as:

$$\mathscr{J}_A(\delta_{x^*} \otimes \delta_z) = e^{x^*(z)} \tag{22}$$

Moreover, we are also in a position to show that the Laplace distributor also "co-transforms" the structural rules into their mirror co-structural rules in the following sense:

▶ **Lemma 7.** *In a differential linear closed category with a Laplace distributor, the following diagrams commute:*



Proof. By Thm 6, we now know that $\ell_A : !A^* \to (!A)^*$ is the unique map such that:

$$(\ell_A \otimes 1_{!A}); \epsilon_{!A,I} = \mu_{A^*,A}; !\epsilon_{A,I}; \mathsf{e}^{\ell} \tag{23}$$

So by using this to our advantage, we can use the same techniques as in the proof of Prop 5 as well as the right side versions of (19), which recall were the compatibility relations between μ and the co-structural maps [2, Prop 2 & Prop 5], to then show that the desired identities hold. Indeed, for $(\ell.\mathsf{d}.2)$, we compute that (omitting subscripts for readability again):

$$(\ell \otimes 1); (\overline{\mathsf{d}}^* \otimes 1); \epsilon \overset{(10)}{=} (\ell \otimes 1); (1 \otimes \overline{\mathsf{d}}); \epsilon = (1 \otimes \overline{\mathsf{d}}); (\ell \otimes 1); \epsilon \overset{(18)}{=} (1 \otimes \overline{\mathsf{d}}); \mu; !\epsilon; \mathsf{e}^{\ell}$$
$$\overset{(19)}{=} (\mathsf{d} \otimes 1); \overline{\mathsf{d}}; !\epsilon; \mathsf{e}^{\ell} \overset{\text{nat.}}{=} (\mathsf{d} \otimes 1); \epsilon; \overline{\mathsf{d}}; \mathsf{e}^{\ell} \overset{(\mathsf{e}.\overline{\mathsf{d}})}{=} (\mathsf{d} \otimes 1); \epsilon$$

So we get that $\ell; \overline{\mathsf{d}}^* = \mathsf{d}$. We can also compute $(\ell.\mathsf{c}.2)$ and $(\ell.\mathsf{w}.2)$ via similar computations. ◀

Using our distribution intuition, we see that these identities for the Laplace distributor correspond again to the three main identities of the exponential function, but expressed in a slightly different manner. So for $y, z \in A$ and $x^* \in A$, we get that:

$$e^{x^*(z+y)} = e^{x^*(z)} e^{x^*(y)} \tag{$\ell.\mathsf{c}.2$}$$
$$e^{x^*(0)} = 1 \tag{$\ell.\mathsf{w}.2$}$$
$$\mathsf{D}_0(e^{x^*(\_)})(x^*(z)) = x^*(z) \tag{$\ell.\mathsf{d}.2$}$$

5 Laplace Transformation

Laplace distributors are also particularly interesting when considered in the *isomix star-autonomous* setting. Recall that a **star-autonomous category** [22, Sec 4.8] is a symmetric monoidal category with a chosen object $\bot$, called the *dualizing object*, such that for every object A, writing $A^{\bot} := A \multimap \bot$, the canonical map $\varrho_A : A \to A^{\bot\bot}$ is an isomorphism. A star-autonomous category whose dualizing object is the monoidal unit $\bot = I$ is called **isomix** [6, Def 6.5]. So in an isomix star-autonomous category, $A^{\bot} = A^*$ and therefore we have the isomorphism $A \cong A^{**}$, and thus every object is *reflexive*. Then by a **differential linear isomix category** we mean a differential linear closed category whose underlying symmetric monoidal closed category is an isomix star-autonomous category. This is a natural setting to consider since many important categorical models of Classical DiLL are isomix.

Now in an isomix star-autonomous category, we can define a new monoidal product $⅋$ defined as $A ⅋ B := (A^* \otimes B^*)^*$ [22, Sec 4.8], where I still acts as a unit for $⅋$. Moreover there is also a canonical natural transformation $\mathsf{m}_{A,B} : A \otimes B \to A ⅋ B$ called the **mixor** [6, Def 6.2] and can be defined as the following composite:

$$A \otimes B \xrightarrow{\rho_A \otimes \rho_B} A^{**} \otimes B^{**} \xrightarrow{\Theta_{A^*,B^*}} (A^* \otimes B^*)^* = A ⅋ B \tag{24}$$

The mixor is the categorical interpretation of the mix rule [13]. Moreover, in a differential linear isomix category $\mathcal{C}$, we can also define the functor $? : \mathcal{C} \to \mathcal{C}$ as $?(\_) = (!\_^*)^*$, and it comes equipped with dual versions of the structural maps of !. So we have natural transformations $\mathsf{p}^?_A : ??A \to ?A$, $\mathsf{d}^?_A : A \to ?A$, $\mathsf{c}^?_A : ?A \parr ?A \to ?A$, $\mathsf{w}^?_A : I \to ?A$, $\mu^?_{A,B} : ?(A \parr B) \to ?A \parr ?B$, $\mu^?_I : ?I \to I$, $\overline{\mathsf{c}}^?_A : ?A \to ?A \parr ?A$, $\overline{\mathsf{w}}^?_A : ?A \to I$, and $\overline{\mathsf{d}}^?_A : ?A \to A$. For example, $\mathsf{d}^?_A : A \to ?A$ and $\overline{\mathsf{d}}^?_A : ?A \to A$ are defined as the following composites:

$$\mathsf{d}^?_A := \; A \xrightarrow{\rho_A} A^{**} \xrightarrow{\mathsf{d}^*_{A^*}} (!A^*)^* = ?A \qquad \overline{\mathsf{d}}^?_A := \; ?A = (!A^*)^* \xrightarrow{\overline{\mathsf{d}}^*_{A^*}} A^{**} \xrightarrow{\rho_A^{-1}} A \tag{25}$$

In particular, this makes $\mathcal{C}^{op}$ a differential linear isomix category as well with monoidal product $\parr$ and with ? its monoidal differential modality.

Now suppose that we have a Laplace distributor $\ell_A : !A^* \to (!A)^*$, and consider its induced Laplace evaluator $\eth_A : !A^* \otimes !A \to I$. Currying the $!A$ gives us back the Laplace distributor. On the other hand, if we curry the $!A^*$ instead, we obtain a map of type $!A \to (!A^*)^* = ?A$, which we call the *Laplace transformation*. Alternatively, the Laplace transformation is the dual of the Laplace distributor, up to the reflexivity isomorphism. Moreover, where the mixor gave a mix rule from the multiplicative conjunction $\otimes$ to the multiplicative disjunction $\parr$, the Laplace transformation provides a mix rule from the exponential conjunction ! to the exponential disjunction ?.

▶ **Definition 8.** *In a differential linear isomix category with a Laplace distributor ℓ, its associated **Laplace transformation** is the natural transformation $\mathcal{L}_A : !A \to ?A$ defined as:*

$$\mathcal{L}_A := \; !A \xrightarrow{\varrho_{!A}} (!A)^{**} \xrightarrow{\ell^*_A} (!A^*)^* = ?A \tag{26}$$

or equivalently, as the unique map such that the following equality holds:

$$(\mathcal{L}_A \otimes 1_{!A^*}); \epsilon_{!A^*,I} = \sigma_{!A,!A^*}; \eth_A \tag{27}$$

Using our distribution intuition, an element of $?A$ is a smooth map $A^* \to I$. Then for every $z \in A$, $\mathcal{L}(\delta_z) : A^* \to I$ is the smooth map defined as:

$$\mathcal{L}(\delta_z)(x^*) = e^{x^*(z)} \tag{28}$$

The induced I-exponential map can also nicely be described in terms of the Laplace transformation:

▶ **Lemma 9.** *For a differential linear isomix category with a Laplace distributor, its induced Laplace transformation is equal to the following composite:*

$$\mathcal{L}_A := \; !A \xrightarrow{!\varrho_A} !A^{**} \xrightarrow{\ell_{A^*}} (!A^*)^* = ?A \tag{29}$$

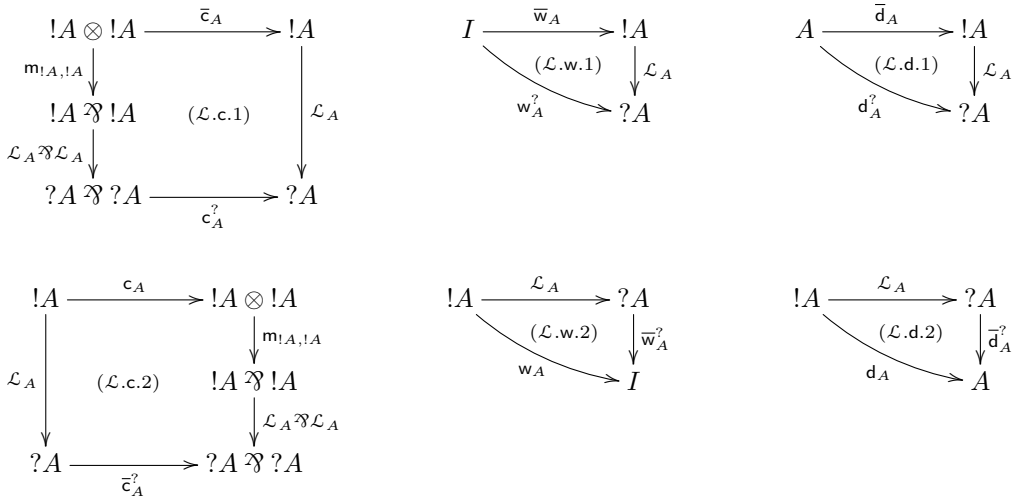
and furthermore, the induced I-exponential map is equal to the following composite:

$$\mathsf{e}^\ell := \; !I \xrightarrow{\mathcal{L}_I} ?I \xrightarrow{\mu^?_I} I \tag{30}$$

Proof. It is straightforward to check that (26) and (29) both satisfy (27), and therefore must be equal. Then (30) follows from the fact that $\mu^?_I = !(\upsilon^*_I); \mu^*_I; \upsilon_I^{-1}$ and that $!\rho_I; !(\upsilon^*_I) = !\upsilon_I$. ◀

Moreover, the analogues of the diagrams of a Laplace distributor in Def 1 and Lemma 7 have nice representations for the Laplace transformation, which is proven easily through the reflexivity isomorphism.

▶ **Proposition 10.** *In a differential linear isomix category with a Laplace distributor, the following diagrams commute:*



Proof. Clearly, ($\mathcal{L}$.c.1), ($\mathcal{L}$.w.1), etc. are precisely the duals (in the sense of applying the contravariant functor $*$) of (ℓ.c.1), (ℓ.w.1), etc. up to the reflexivity isomorphism. ◀

So we clearly see how the Laplace transformation does indeed transform the co-structural rules of ! into the structural rules of ?, where the latter are the dual of the structural rules of !. Moreover, we note that ($\mathcal{L}$.c.1 & 2) and ($\mathcal{L}$.w.1 & 2) say that the Laplace transformation is a morphism from a $\otimes$-(co)monoid to a $\parr$-(co)monoid. This is a key idea for the exponential modality in *dagger* linear logic [8].

Another natural question to ask is if from a Laplace distributor ℓ, we can get isomix star-autonomy. To answer this, note that the functor ? can be defined for any symmetric monoidal closed category. However, ? will not have all the structural maps above since $\parr$ is no longer necessarily a monoidal product. Nevertheless, for any differential linear closed category with a Laplace distributor, we can always get a natural transformation of type $\mathcal{L}_A : !A \to ?A$. If there is also a map of dual type which agrees on the A parts of $!A$ and $?A$ then we do get isomix star-autonomy. This is particularly the case when $\mathcal{L}_A : !A \to ?A$ is an isomorphism. In calculus, that the Laplace transform is reversible is particularly important: this makes it a tool to go from the differential world to the polynomial world and back.

▶ **Proposition 11.** *Let $\mathcal{C}$ be a differential linear closed category with a Laplace distributor ℓ and induced natural transformation $\mathcal{L}_A : !A \to ?A$ as defined in (29). If there is a natural transformation $\mathcal{L}^\bullet : ?A \to !A$ such that the following equalities hold:*

$$\bar{\mathsf{d}}_A ; \mathcal{L}_A ; \mathcal{L}^\bullet_A ; \mathsf{d}_A = 1_A \qquad\qquad \mathsf{d}^*_{A^*} ; \mathcal{L}^\bullet_A ; \mathcal{L}_A ; \bar{\mathsf{d}}^*_{A^*} = 1_{A^{**}} \tag{31}$$

then $\mathcal{C}$ is isomix star-autonomous. In particular, if $\mathcal{L}$ is an isomorphism, then $\mathcal{C}$ is isomix star-autonomous.

Proof. The key to this proof is that by definition of the co-dereliction [2, Def 9], we have that $\bar{\mathsf{d}} ; \mathsf{d} = 1$. From this, naturality, ($\mathcal{L}$.d.1), and (ℓ.d.1), we get that $\rho = \bar{\mathsf{d}} ; \mathcal{L} ; \bar{\mathsf{d}}^*$. Then define $\rho^{-1} = \mathsf{d}^* ; \mathcal{L}^\bullet ; \mathsf{d}$. Then from (31) and ($\ell$.d.1 & 2), one easily checks that $\rho ; \rho^{-1} = 1$ and $\rho^{-1} ; \rho = 1$. So we conclude that $\mathcal{C}$ is isomix star-autonomous as desired. Now if $\mathcal{L}$ was an isomorphism to begin with, setting $\mathcal{L}^\bullet = \mathcal{L}^{-1}$, it follows from ($\ell$.d.1 & 2) that (31) holds. ◀

6 Examples

In this section, we give examples of Laplace distributors/evaluators, I-exponential maps, and Laplace transformations in well-known and important examples of differential categories.

▶ **Example 12** (Relations)**.** An important model of DiLL is the relational model. So let REL be the category whose objects are sets X and where a map $R : X \to Y$ is a relation, that is, a subset $R \subseteq X \times Y$. REL is a differential linear isomix category; for full details, see [18, Sec IV.A]. In particular, for a set X, $!X = \mathfrak{M}_f(X)$ is the set of finite multisets of X. Moreover, the monoidal unit is a chosen singleton $I = \{*\}$, and (up to isomorphism) we may associate $X^* = X$. From this point of view $?X = !X$, and we trivially see that the identity:

$$1_{!X} = \{(B, B) | \forall B \in !X\} \subseteq !X \times !X$$

is a Laplace distributor and its induced Laplace transformation. Moreover, the induced $\{*\}$-exponential map is the one that relates every bag to the single element, in other words:

$$\mathsf{e} = !\{*\} \times \{*\}$$

REL is also an example of both the more general settings described in Ex 15 and Ex 16 below.

▶ **Example 13** (Weighted Relations)**.** The relation model can be generalized by considering *weighted* relations [23, Sec III] over a *complete commutative semiring*. Recall that a commutative semiring R is complete if sums of elements of R indexed by arbitrary sets are well-defined in R, and these sums satisfy natural distributivity and partition axioms [23, Sec III.C]. For a complete commutative semiring R, define the category R^{Π} whose objects are sets X, and where a map from X to Y is a function $f : X \times Y \to R$. Composition of $f : X \times Y \to R$ and $g : Y \times Z \to R$ is defined as $(f; g)(x, z) = \sum_{y \in Y} f(x, y)g(y, z)$, which is well-defined since R is complete. The identity is the Kronecker delta function $\delta_X : X \times X \to R$ defined as $\delta_X(x, y) = 0$ if $x \neq y$ and $\delta_X(x, x) = 1$. Then R^{Π} is a differential linear isomix category; for full details, see [18, Sec IV.B]. In particular, as in Ex 12, the modality is $!X = \mathfrak{M}_f(X)$, the monoidal product is the Cartesian product of sets $\times$ (which is not the categorical product), and the monoidal unit is $I = \{*\}$. However, in general, R^{Π} will not have an $\{*\}$-exponential map. Suppose that we do have an $\{*\}$-exponential map, so a function $\mathsf{e} : !\{*\} \times \{*\} \to R$. First note that elements of $!\{*\}$ can be associated with the natural numbers: so for every $n \in \mathbb{N}$, let $[n]$ be the finite multiset with n copies of $*$. Now the co-dereliction is:

$$\overline{\mathsf{d}}_{\{*\}}(*, [n]) = \delta_{\mathbb{N}}(1, n)$$

Then $(\mathsf{e}.\overline{\mathsf{d}})$ would give us that: $\mathsf{e}([1], *) = 1$. On the other hand, the co-contraction is: $\overline{\mathsf{c}}_{\{*\}}(([n], [m]), [k]) = \binom{n+m}{m}\delta_{\mathbb{N}}(n + m, k)$. So $(\mathsf{e}.\overline{\mathsf{c}})$ would give us that:

$$\binom{n+m}{m}\mathsf{e}([n + m], *) = \mathsf{e}([n], *)\mathsf{e}([m], *)$$

Now, taking $n = m = 1$ in this last equality, we would get that $2\mathsf{e}([2], *) = 1$, which says that $\mathsf{e}([2], *)$ is an inverse of 2 in R. However, 2 is not always a unit in an arbitrary semiring. For example, $R = \mathbb{N} \sqcup \{\infty\}$ is a complete commutative semiring for which 2 is not invertible. Therefore, $(\mathbb{N} \sqcup \{\infty\})^{\Pi}$ is a differential linear closed/isomix category which does not have an $\{*\}$-exponential map. Now if every n is invertible in R, then R^{Π} does have a $\{*\}$-exponential map given by the function $\mathsf{e} : !\{*\} \times \{*\} \to R$ defined as: $\mathsf{e}([n], *) = \frac{1}{n!}$. That is indeed an

$\{*\}$-exponential map follows from the fact that R^{Π} is an example of the general settings described in Ex 15 and Ex 16. Moreover, this example also recaptures Ex 12 since by taking the Boolean semiring $\mathbb{B} = \{0, 1\}$, we get back $\mathbb{B}^{\Pi} \cong \mathsf{REL}$, and since $1 + 1 = 1$ in $\mathbb{B}$, the factor $\frac{1}{n!}$ disappears in the descriptions of the $\{*\}$-exponential map in REL.

▶ **Example 14** (Convenient Vector Spaces)**.** Throughout the paper, we used distributions for intuition. We can make this precise by considering the differential category of *convenient spaces*, introduced by Blute, Ehrhard, and Tasson in [1]. Briefly, a **convenient vector space** [1, Def 2.9] is a special kind of locally convex vector space which in particular has a bornology for which it is Mackey complete. This allows us to define smooth functions (in the usual analysis sense) between convenient vector spaces E and F. Let $\mathcal{C}^{\infty}(E, F)$ be the set of smooth functions between them, which is itself a convenient vector space. Also, the reals $\mathbb{R}$ is a convenient vector space, and for a convenient vector space E, we let E^* be the vector space of *linear* smooth functions $E \to \mathbb{R}$, which is again a convenient vector space. Now, for every convenient vector space E, we have a smooth function $\delta : E \to \mathcal{C}^{\infty}(E, \mathbb{R})^*$ which maps $x \in E$ to its associated Dirac distribution $\delta_x \in \mathcal{C}^{\infty}(E, \mathbb{R})^*$. Then CON, the category of convenient vector spaces and linear smooth functions between them, is a differential linear closed category, where $!E$ is the Mackey completion of $\delta(E) \subseteq \mathcal{C}^{\infty}(E, \mathbb{R})^*$ [1, Def 5.2]. Moreover, for every smooth function $f : E \to F$ there is a unique linear smooth function $f^{\sharp} : !E \to F$ such that $f(x) = f^{\sharp}(\delta_x)$ [1, Thm 5.5]. Now the monoidal unit in CON is $\mathbb{R}$ and since the classical exponential function $e^z : \mathbb{R} \to \mathbb{R}$ is smooth, there exists a unique linear smooth function $\mathsf{e} : !\mathbb{R} \to \mathbb{R}$ such that:

$$e^z = \mathsf{e}(\delta_z)$$

From this, it immediately follows that $\mathsf{e} : !\mathbb{R} \to \mathbb{R}$ is an $\mathbb{R}$-exponential map. Therefore, the Laplace distributor and Laplace evaluator are precisely the unique linear smooth functions such that:

$$\ell_E(\delta_{x^*}) = e^{x^*(-)} \qquad \qquad \jmath_A(\delta_{x^*} \otimes \delta_z) = e^{x^*(z)}$$

As such, this is the model which properly interprets the Laplace transform as operating on Dirac distributions as discussed in Section 3.

▶ **Example 15** (Countable Sums)**.** In calculus, the exponential function can be written out as the power series:

$$e^x = \sum_{n \in \mathbb{N}} \frac{x^n}{n!}$$

In a setting where we have countable sums and can scalar multiply by positive rationals $\mathbb{Q}_{\geq 0}$, the same formula holds for constructing an I-exponential map. So suppose that we are in a $\mathbb{Q}^{\Sigma}_{\geq 0}$**-differential linear closed/isomix category** as was considered in [18, Sec III.E], which means that each homset is a countably complete $\mathbb{Q}_{\geq 0}$-module. Now for every $n \in \mathbb{N}$, let $\mathsf{c}^n_A : !A \to !A^{\otimes^n}$ be the map which co-multiplies $!A$ into n-copies of $!A$, and then define $\mathsf{d}^n_A : !A \to A^{\otimes^n}$ as the composite:

$$\mathsf{d}^n_A := \; !A \xrightarrow{\mathsf{c}^n_A} !A \otimes \ldots \otimes !A \xrightarrow{\mathsf{d}_A \otimes \ldots \otimes \mathsf{d}_A} A \otimes \ldots \otimes A$$

With these maps, we can construct an I-exponential map defined as follows:

$$\mathsf{e} := \sum_{n=0}^{\infty} \frac{1}{n!} \cdot \left(!I \xrightarrow{\mathsf{d}^n_I} I^{\otimes^n} \xrightarrow{\cong} I \right)$$

Checking that this is an I-exponential map is the same proof that checking that e^x satisfies the analogue identities using its power series. Then, the induced Laplace distributor, Laplace evaluator, and Laplace transformation are given as follows:

$$\ell_A := \sum_{n=0}^{\infty} \frac{1}{n!} \cdot \left(!A^* \xrightarrow{\mathsf{d}^n_{A^*}} A^* \otimes \ldots \otimes A^* \xrightarrow{\Theta_{A,\ldots,A}} (A \otimes \ldots \otimes A)^* \xrightarrow{\mathsf{d}^{n\,*}_A} (!A)^* \right)$$

$$\gimel_A := \sum_{n=0}^{\infty} \frac{1}{n!} \cdot \left(!A^* \otimes !A \xrightarrow{\mathsf{d}^n_{A^*} \otimes \mathsf{d}^n_A} (A^*)^{\otimes^n} \otimes A^{\otimes^n} \xrightarrow{\cong} (A^* \otimes A)^{\otimes^n} \xrightarrow{\epsilon^{\otimes^n}_{A,I}} I^{\otimes^n} = I \right)$$

$$\mathcal{L}_A := \sum_{n=0}^{\infty} \frac{1}{n!} \cdot \left(!A \xrightarrow{\mathsf{d}^n_A} A \otimes \ldots \otimes A \xrightarrow{\mathsf{m}_{A,\ldots,A}} A \parr \ldots \parr A \xrightarrow{\mathsf{d}^{?\,n}_A} ?A \right)$$

▶ **Example 16** (Co-digging)**.** In [18], the authors introduced the notion of *co-digging* for differential categories, which is the co-structural version of digging. Briefly, **co-digging** [18, Def III.3] for a differential linear category is a natural transformation of dual type of the digging, $\overline{\mathsf{p}}_A : !!A \to !A$, which satisfies the dual axioms of the digging. Using our distribution intuition, the co-digging corresponds to the notion of convolution exponential for distributions:

$$\overline{\mathsf{p}}_A(\delta_{\delta_z}) = \sum_{n \in \mathbb{N}} \frac{\delta_{nx}}{n!} \tag{32}$$

See [18, Sec III.C] for more details. The co-digging always induces an I-exponential map $\overline{\mu}_I : !I \to I$ [18, Lemma III.4] defined as the composite:

$$\overline{\mu}_I := \; !I \xrightarrow{!\overline{\mathsf{w}}_I} !!I \xrightarrow{\overline{\mathsf{p}}_I} !I \xrightarrow{\mathsf{w}_I} I \tag{33}$$

Therefore, every differential linear closed/isomix category with co-digging has a Laplace distributor/transformation. Furthermore, if $\overline{\mathsf{p}}$ and μ are compatible in the sense that the following diagram commutes:

$$\begin{array}{ccccccc} !!A \otimes !B & \xrightarrow{1_{!A} \otimes \mathsf{p}_B} & !!A \otimes !!B & \xrightarrow{\mu_{!A,!B}} & !\,(!A \otimes !B) & \xrightarrow{!(\mu_{A,B})} & !!(A \otimes B) \\ \downarrow{\scriptstyle \overline{\mathsf{p}}_A \otimes 1_{!B}} & & & & & & \downarrow{\scriptstyle \overline{\mathsf{p}}_{!A,!B}} \\ !A \otimes !B & & \xrightarrow[\mu_{A,B}]{} & & & & A \otimes B \end{array} \tag{34}$$

which is the same kind of compatiblity as those in (19) – then using the same techniques as in the proof of Prop 5, we also get that the Laplace distributor/transformation also (co-)transforms the co-digging into the digging:

$$\begin{array}{ccc} !!A^* & \xrightarrow{\overline{\mathsf{p}}_{A^*}} & !A^* \\ \downarrow{\scriptstyle !(\ell_A)} & & \\ !(!A)^* \;(\ell.\mathsf{p}.1) & & \downarrow{\scriptstyle \ell_A} \\ \downarrow{\scriptstyle \ell_{!A}} & & \\ (!!A)^* & \xrightarrow[\mathsf{p}^*_A]{} & (!A)^* \end{array} \qquad \begin{array}{ccc} !A^* & \xrightarrow{\ell_A} & (!A)^* \\ \downarrow{\scriptstyle \mathsf{p}_{A^*}} & & \\ !!A^* \quad (\ell.\mathsf{p}.2) & & \downarrow{\scriptstyle \overline{\mathsf{p}}^*_A} \\ \downarrow{\scriptstyle !(\ell_A)} & & \\ !(!A)^* & \xrightarrow[\ell_{!A}]{} & (!!A)^* \end{array} \qquad \begin{array}{ccc} !!A & \xrightarrow{\overline{\mathsf{p}}_A} & !A \\ \downarrow{\scriptstyle !(\mathcal{L}_A)} & & \\ !?A \;(\mathcal{L}.\mathsf{p}.1) & & \downarrow{\scriptstyle \mathcal{L}_A} \\ \downarrow{\scriptstyle \mathcal{L}_{?A}} & & \\ ??A & \xrightarrow[\mathsf{p}^?_A]{} & ?A \end{array} \qquad \begin{array}{ccc} !A & \xrightarrow{\mathcal{L}} & ?A \\ \downarrow{\scriptstyle \mathsf{p}_A} & & \\ !!A \;(\mathcal{L}.\mathsf{p}.2) & & \downarrow{\scriptstyle \overline{\mathsf{p}}^?_A} \\ \downarrow{\scriptstyle !(\mathcal{L}_A)} & & \\ !?A & \xrightarrow[\mathcal{L}_{?A}]{} & ??A \end{array}$$

These diagrams amount to interpreting the exponential function of the exponential function, e^{e^z}. So, in particular, in terms of the Laplace distributor:

$$e^{e^{x^*(z)}} = \sum_{n\in\mathbb{N}} \frac{e^{nx^*(z)}}{n!} \tag{ℓ.p.1}$$

$$e^{e^{x^*(z)}} = \sum_{n\in\mathbb{N}} \frac{e^{x^*(nz)}}{n!} \tag{ℓ.p.2}$$

Interesting examples of models with co-digging can be found in [18, Sec IV], which include the (weighted) relational model described above.

▶ **Example 17** (Finiteness Spaces)**.** Finiteness spaces [10] are a well-known refinement of REL, giving a vectorial model of DiLL [11]. For a set X, and a subset of its powerset $F \subseteq \mathcal{P}(X)$, we denote by $F^\perp \subseteq \mathcal{P}(X)$ the subsets of $U \subseteq X$ such that for all $V \in F$, $U \cap V$ is finite. Then a **finiteness space** [10, Sec 1] is a pair $\mathcal{X} = (|\mathcal{X}|, \mathfrak{F}(\mathcal{X}))$ of a set $|\mathcal{X}|$ and a subset $\mathfrak{F}(\mathcal{X}) \subseteq \mathcal{P}(|\mathcal{X}|)$ which verifies the finiteness condition $\mathfrak{F}(\mathcal{X})^{\perp\perp} = \mathfrak{F}(\mathcal{X})$. Given a field k, every finiteness space $\mathcal{X}$ generates a linear topological k-vector space $k\langle\mathcal{X}\rangle$ defined as the set of all families $x \in k^{|\mathcal{X}|}$ such that $\mathsf{supp}(x) = \{a \in |X| | x_a \neq 0\} \in \mathfrak{F}(X)$ [10, Sec 3]. Then let FIN^k be the category whose objects are finiteness spaces and where a map from $\mathcal{X}$ to $\mathcal{Y}$ is a linear continuous function $M : k\langle\mathcal{X}\rangle \to k\langle\mathcal{Y}\rangle$, which can be described as an $|\mathcal{X}| \times |\mathcal{Y}|$ matrix $M_{x,y}$. Then FIN^k is a differential linear isomix category; see [11, Sec 5] for full details. The finiteness space $!\mathcal{X}$ has as carrier $|!\mathcal{X}| := \mathfrak{M}_f(|\mathcal{X}|)$, the finite multisets over $|\mathcal{X}|$, and as finiteness structure $\mathfrak{F}(!\mathcal{X}) \subseteq \mathcal{P}(\mathfrak{M}_f(|\mathcal{X}|))$ the collection of all sets of multisets on $|X|$ whose union is in $\mathfrak{F}(\mathcal{X})$ [10, Sec 1.1]. On the other hand, the monoidal unit is $I = (\{*\}, \mathcal{P}(*))$. Then FIN^k has an I-exponential map given by the exponential function described by Ehrhard in [10, Lemma 19], that is, the linear continuous function $\mathsf{e} : k\langle!I\rangle \to k\langle I\rangle$ whose associated matrix is:

$$e_{n,*} = \frac{1}{n!}$$

Now for a finiteness space $\mathcal{X}$, its dual is the finiteness space $\mathcal{X}^* = (|\mathcal{X}|, \mathfrak{F}(\mathcal{X})^\perp)$. Then both the induced Laplace distributor $\ell_\mathcal{X} : k\langle!\mathcal{X}^*\rangle \to k\langle(!X)^*\rangle$ and Laplace transformation $\mathcal{L}_\mathcal{X} : k\langle!X\rangle \to k\langle?\mathcal{X}\rangle$ have the same associated matrix with coefficients indexed by multisets $m, m' \in \mathfrak{M}_f(|\mathcal{X}|)$:

$$(\ell_\mathcal{X})_{m,m'} = (\mathcal{L}_\mathcal{X})_{m,m'} = e^{m'(m)} = e^{\sum_{x\in|\mathcal{X}|} m(x)m'(x)}$$

which is well-defined thanks to the orthogonality condition.

▶ **Example 18** (Köthe Spaces)**.** Köthe spaces [9] are a model of DiLL based on spaces of sequences. They are studied independently in functional analysis and correspond to a non-discrete version of finiteness spaces. Let k be the field of real or complex numbers. For a denumerable set X, for sequences $a, b \in k^X$ define the orthogonality relation $a \perp b$ if and only if $\sum_{x\in X} |a_x b_x|$ converges. A Köthe space is pair $\mathcal{X} = (|\mathcal{X}|, E_\mathcal{X})$ of a carrier set $|\mathcal{X}|$ and a subspace $E_\mathcal{X} \subseteq k^{|\mathcal{X}|}$, such that $E_\mathcal{X}^{\perp\perp} = E_X$. Then we get a differential linear isomix category of Köthe spaces, which is similar to the finiteness space model described above. In particular, the exponential function corresponds to taking a converging sequence $a \in k^\mathbb{N}$ to $\sum_{n\in\mathbb{N}} \frac{1}{n!} a_n$. The Laplace distributor/transformation is expressed similarly to finiteness spaces.

▶ **Example 19** (Fréchet and DF spaces)**.** Fréchet spaces are metrizable and complete locally convex topological vector spaces. They enjoy a nice duality theory with DF-spaces. When adding the constraint that these spaces must be nuclear [16], one obtains a model of polarized

first-order DiLL [17]: the tensor product of two nuclear DF-space is a nuclear DF-space; nuclear DF or Fréchet spaces are isomorphic to their double duals; $?\mathbb{R}^n = \mathcal{C}^\infty(\mathbb{R}^n, \mathbb{R})$ is nuclear DF; and $!\mathbb{R}^n = \mathcal{C}^\infty(\mathbb{R}^n, \mathbb{R})'$ is nuclear Fréchet. A nuclear Fréchet spaces N is, in fact, a projective limit of Banach Spaces $N = \bigcap_n N_p$. This construction is taken to the higher-order level in [14], using functions whose exponential growth is bounded. For a Young function θ and for a Banach space B, let $Exp(B, \theta, m)$ denote the Banach space of holomorphic functions from B to $\mathbb{C}$ such that $|f(z)| \leq Ke^{\theta(m||z||)}$. Then, one defines the space of functions with exponential growth of minimal order on N as the inductive limit $\mathcal{G}_{\theta^*}(N)$ of spaces $\mathrm{Exp}(\theta, m, p)$, and also the space of functions with exponential growth of arbitrary order on $N' = \bigcup_p (N_p)' = \bigcup_p N'_{-p}$ as the projective limit $\mathcal{F}_\theta(N')$ of spaces $\mathrm{Exp}(\theta, m, -p)$. In this higher-order setting, the Laplace transform has a finer meaning than in the other examples. Indeed, it transforms distributions on one type of function into another type of function and makes the index θ change:

$$\mathcal{L} : \left\{ \begin{array}{ccc} \mathcal{F}'_\theta(N') & \simeq & \mathcal{G}_{\theta^*}(N) \\ \phi & \mapsto & \big(\ell \in N' \mapsto \phi(x \in N' \mapsto e^{\ell(x)} \in \mathbb{C})\big) \end{array} \right.$$

where $\theta^* := sup_{t \geq 0}(tx - \theta(t))$ is the convex conjugate of θ. Details about this construction can be found in [18, Sec V]. This opens up fascinating questions on Laplace transforms in polarized differential linear categories [4] or even graded differential linear categories [21].

7 Future Work

In this paper, we gave a new point of view on exponential functions in differential categories, and on the exponential connectives ! and ? in DiLL, thanks to the categorification of the Laplace transform. We defined the Laplace distributor as a transformation from the exponential $!A$ to its dual which transforms co-structural rules into structural rules. We related this new distributor to the presence of an exponential scalar function, and to the involutivity of the duality. We presented several examples, as well as one counter-example. We conclude this paper with a brief discussion of interesting potential future work.

A natural path to consider is generalizing this story from isomix star-autonomous categories to *linearly distributive categories* [5]. Indeed, the diagrams in Prop 10 can easily be written down in a linearly distributive category with the proper notion of exponentials [7, 8]. So one could study Laplace transformations in a linearly distributive setting. However, the linearly distributive generalization of differential categories has not yet been properly defined or studied. So hopefully the story of this paper will motivate the development of such a theory.

Work is also needed on concrete models of Laplace transforms. The original intuition for the categorification of the Laplace transform came from higher-order work in functional analysis [14, 18], in which two kinds of functions with different exponential growth model the two types of exponential connectives, applying to formulas with different polarities [19]. The Laplace transformation then changes distributions on one type of function into distributions on the other type of function. Understanding the categorical interplay between the Laplace transformation and polarity might lead to a better axiomatization of differential linear star-autonomous linear categories.

References

1 R. Blute, T. Ehrhard, and C. Tasson. A convenient differential category. *Cahiers de topologie et géométrie différentielle catégoriques*, 2012.

2 R. F. Blute, J. R. B. Cockett, J.-S. P. Lemay, and R. A. G. Seely. Differential categories revisited. *Applied Categorical Structures*, 2020.

3 R. F. Blute, J. R. B. Cockett, and R. A. G. Seely. Differential categories. *Math. Struct. Comput. Sci.*, 2006.

4 F. Breuvart, M. Kerjean, and S. Mirwasser. Unifying graded linear logic and differential operators. In *8th International Conference on Formal Structures for Computation and Deduction, FSCD*, 2023.

5 J. R. B. Cockett and R. A. G. Seely. Weakly distributive categories. *Journal of Pure and Applied Algebra*, 114(2), 1997.

6 J. R. B. Cockett and R.A.G. Seely. Proof theory for full intuitionistic linear logic, bilinear logic, and mix categories. *Theory and Applications of categories*, 3(5), 1997.

7 J. R. B. Cockett and R.A.G. Seely. Linearly distributive functors. *Journal of Pure and Applied Algebra*, 143(1-3), 1999.

8 J.R.B. Cockett and P. V. Srinivasan. Exponential modalities and complementarity (extended abstract). In Kohei Kishida, editor, Proceedings of the Fourth International Conference on *Applied Category Theory,* Cambridge, United Kingdom, 12-16th July 2021, volume 372 of *Electronic Proceedings in Theoretical Computer Science*. Open Publishing Association, 2022.

9 T. Ehrhard. On Köthe Sequence Spaces and Linear Logic. *Mathematical Structures in Computer Science*, 12(5), 2002.

10 T. Ehrhard. Finiteness spaces. *Mathematical Structures in Computer Science*, 15(4), 2005.

11 T. Ehrhard. An introduction to differential linear logic: proof-nets, models and antiderivatives. *Mathematical Structures in Computer Science*, 2017.

12 T. Ehrhard and L. Regnier. Differential interaction nets. *Theoretical Computer Science*, 364(2), 2006.

13 A. Fleury and C. Retoré. The mix rule. *Mathematical Structures in Computer Science*, 4(2), 1994.

14 R. Gannoun, R. Hachaichi, H. Ouerdiane, and A. Rezgui. Un théorème de dualité entre espaces de fonctions holomorphes à croissance exponentielle. *Journal of Functional Analysis*, 171(1), 2000.

15 J.-Y. Girard. Linear Logic. *Theoretical Computer Science*, 50(1), 1987.

16 A. Grothendieck. Produits tensoriels topologiques et espaces nucléaires. *Memoirs of the AMS*, 16, 1966. Publisher: American Mathematical Society.

17 M. Kerjean. A Logical Account for Linear Partial Differential Equations. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, 2018.

18 M. Kerjean and J.-S. P. Lemay. Taylor Expansion as a Monad in Models of DiLL. In *2023 38th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 2023.

19 O. Laurent. *Etude de la polarisation en logique*. Thèse de Doctorat, Université Aix-Marseille II, March 2002.

20 J.-S. P. Lemay. Exponential functions in cartesian differential categories. *Applied Categorical Structures*, 29, 2021.

21 J.-S. P. Lemay and J.-B. Vienney. Graded differential categories and graded differential linear logic. In *Mathematical Foundations of Programming Semantics*, 2023.

22 P.-A. Melliès. Categorical semantics of linear logic. *Société Mathématique de France*, 2008.

23 C.-H. L. Ong. Quantitative semantics of the lambda calculus: Some generalisations of the relational model. In *31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 2017.

Second-Order Generalised Algebraic Theories: Signatures and First-Order Semantics

Ambrus Kaposi
Eötvös Loránd University, Budapest, Hungary

Szumi Xie
Eötvös Loránd University, Budapest, Hungary

Abstract

Programming languages can be defined from the concrete to the abstract by abstract syntax trees, well-scoped syntax, well-typed (intrinsic) syntax, algebraic syntax (well-typed syntax quotiented by conversion). Another aspect is the representation of binding structure for which nominal approaches, De Bruijn indices/levels and higher order abstract syntax (HOAS) are available. In HOAS, binders are given by the function space of an internal language of presheaves. In this paper, we show how to combine the algebraic approach with the HOAS approach: following Uemura, we define languages as second-order generalised algebraic theories (SOGATs). Through a series of examples we show that non-substructural languages can be naturally defined as SOGATs. We give a formal definition of SOGAT signatures (using the syntax of a particular SOGAT) and define two translations from SOGAT signatures to GAT signatures (signatures for quotient inductive-inductive types), based on parallel and single substitutions, respectively.

2012 ACM Subject Classification Theory of computation → Type theory

Keywords and phrases Type theory, universal algebra, inductive types, quotient inductive types, higher-order abstract syntax, logical framework

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.10

Supplementary Material *Software (Implementation and detailed constructions)*: `https://bitbucket.org/akaposi/sogat`, archived at `swh:1:dir:0f9fefbd89f1151eedb46518bde3062c0bc5a3fe`

Funding *Ambrus Kaposi*: The author was supported by project no. TKP2021-NVA-29 which has been implemented with the support provided by the Ministry of Culture and Innovation of Hungary from the National Research, Development and Innovation Fund, financed under the TKP2021-NVA funding scheme.

1 Introduction

The traditional way of defining a programming language comprises of a BNF-style description of abstract syntax trees, a typing relation and a reduction or conversion relation [48, 49, 53]. If instead the syntax is defined using well-scoped syntax trees [34, 27, 3], bound names do not matter: for example, one cannot distinguish $\lambda x.x$ and $\lambda y.y$ anymore. A higher level representation is given by intrinsic (well-typed) terms [9, 53] where one merges the syntax and the typing relation: non well-typed terms are not expressable in such a representation. The next level of abstraction is when well-typed terms are quotiented by the conversion relation: this is especially convenient for dependently typed languages where typing depends on conversion [7]. Here one can only define functions on the syntax that preserve conversion: a simple printing function is not definable, but normalisation [6, 20], typechecking [35] or parametricity [7] preserve conversion, so they can be defined on the well-typed quotiented syntax. The well-typed quotiented syntax is also concordant with the semantics: there is no reason to have a separate definition of syntax and a different notion of semantics, but the syntax can be simply defined as the initial model, which always exists for any generalised algebraic theory (GAT) [39]. Thus, abstractly, a language is simply a GAT.

Another aspect of the definition of a language is the treatment of bindings and variables: one can use De Bruijn indices to make sure that choices of names do not matter, but then substitution has to be part of the syntax, for example in the form of a category with families [19]. Logical frameworks [29, 47] and higher-order abstract syntax (HOAS) [32] provide another way to implement bindings and variables: they use the function space of the metatheory. For example, the type of the lambda operation in the pure lambda calculus is simply the second-order function space $(\mathsf{Tm} \to \mathsf{Tm}) \to \mathsf{Tm}$. The justification of HOAS is the type-theoretic internal language of presheaves over the category of contexts and syntactic substitutions [32]. In this internal language, lambda indeed has the above type. This internal language viewpoint can also be used to *define* languages: in this case a language with bindings is not a GAT, but a second-order generalised algebraic theory (SOGAT), which allows second-order (but not general higher-order) operations. While untyped or simply typed languages were defined as second-order theories before [23, 21, 2], SOGATs were first used by Uemura [52] for defining languages with bindings. The step from second-order algebraic theories to SOGATs is a big one: it is analogous to the step from inductive types to inductive-inductive types [40]. The SOGAT definition of a language can be even more abstract than the well-typed quotiented definition: the SOGAT does not mention contexts or substitutions: these can be seen as boilerplate that should be automatically generated. SOGATs are not well-behaved algebraic theories, for example, there is no meaningful notion of homomorphism of second-order models. To describe first order models, homomorphisms or the notion of syntax for a SOGAT, we turn it into a GAT. In this process we introduce new sorts for contexts and substitutions, we index every operation with its context, and the second-order function spaces become first order using this context indexing. The thus obtained GAT has some "correctness by construction" properties, for example, every operation automatically preserves substitution. For complicated theories, this property is not trivial if we do not start from a SOGAT, but try to work with the lower level GAT presentation directly.

Cubical type theory [51] and a type theory with internal parametricity [5] have been presented as SOGATs, and methods were developed to prove properties of type theories at the SOGAT level of abstraction [50, 16]. Substructural (e.g. linear or modal) type theories are not definable as SOGATs using the method described in this paper, but sometimes presheaves over a substructural theory provide a substructural internal language which can be used to describe the theory, as in the case of multi-modal type theory [26].

Simple algebraic theories can be presented using signatures and equations, or presentation-independently as Lawvere theories. GATs have syntactic signatures defined using preterms and well-formedness relations [18], and they can be described presentation-independently as contextual categories [18], categories with families (CwFs) or clans [24]. The "theory of signatures" (ToS) approach [39] is halfway between the syntactic and presentation-independent approaches: here signatures are defined by the syntax of a particular GAT, which is a domain-specific type theory designed for defining signatures. Signatures look exactly as we write inductive datatype definitions in a proof assistant like Agda: a list (telescope) of the curried types of sorts and constructors. A signature in the ToS is a concrete presentation of a theory, but it is given at the level of abstraction of well-typed quotiented syntax. This allows elegant semantic constructions [43], while still working directly with signatures. SOGATs again can be defined syntactically [52] or presentation-independently as representable map categories [52] or CwFs with locally representable types [14]. The current paper contributes the ToS style definition of SOGATs (we leave the proof of equivalence with the former definitions as future work). The theory of SOGAT signatures is itself a SOGAT which can describe itself. Circularity is avoided because we bootstrap the theory of SOGAT signatures by first defining it as a GAT, and the theory of GAT signatures (which is the syntax of a GAT) can itself be bootstrapped using a Church-encoding [42].

Contributions. The main takeaway of this paper is that structural languages are SOGATs. We justify this claim through several examples. Our technical contributions are the following:

- The theory of SOGAT signatures (ToS^+), a domain-specific type theory in which every closed type is a SOGAT signature. As it is a structural type theory, it can be defined as a SOGAT itself. Signatures can be formalised in ToS^+ without encoding overhead.
- A translation from SOGAT signatures to GAT signatures based on a parallel substitution calculus. Thus, for every SOGAT, we obtain all of the semantics of GATs: a category of models with an initial object, (co)free models, notions of displayed models and sections, the fact that induction is equivalent to initiality, and so on. The GAT descriptions that we obtain are readable, do not contain occurrences of Yoneda as in usual presheaf function spaces. Correctness of the translation is showed by proving that internally to presheaves over a model of the GAT, a second-order model of the SOGAT is available.
- We define an alternative translation producing a single substitution calculus.

Structure of the paper. In Section 2, we walk through examples of languages defined as second-order algebraic theories (SOGATs) including (simply typed) combinator calculus, (simply typed) lambda calculus, first-order logic, System $\mathrm{F}(\omega)$, Martin-Löf type theory. We list more examples in Appendix A including the lambda cube. We explain what the SOGAT $\to$ GAT translation will give for each example. In Section 3, we define languages for describing algebraic theories, culminating in the theory of SOGAT signatures (ToS^+). A SOGAT is simply a closed type in the syntax of ToS^+. Then we define the SOGAT $\to$ GAT translation in three iterations: Section 4 presents a naive notion of model which is obviously correct, but has lots of encoding overhead. Section 5 defines an isomorphic notion of model with less encoding overhead. The final translation is defined in Section 6. Section 7 discusses open and infinitary signatures, and explains the single substitution calculus variant. Section 8 concludes.

Related work. The "theory of signatures" (ToS) approach was introduced by Kaposi and Kovács [38] for a higher variant of GATs (higher inductive-inductive types), and was used to describe ordinary [39] and infinitary [42] GATs (quotient inductive-inductive types). The thesis of Kovács [43] summarises and generalises these results, in particular, it provides semantics internal to any category with families (CwF) using the semantic setting of two-level type theory [4, 10]. The current paper extends this work with second-order operations. The ToS that we use differs from the one in Kovács' thesis by including Σ types and being presented as a SOGAT itself. This has the advantage that we do not have to deal with De Bruijn indices when giving formal signatures. A version of ToS^+ with two fixed sorts of types and terms was given in the HoTTeST talk by Kaposi [36].

Direct precursors of our work are Hofmann's analysis of higher-order abstract syntax (HOAS) [32] and Capriotti's rule framework [17]. Syntactic definitions of SOGATs are given in Uemura's thesis [52] and Harper's equational logical framework [28]. A syntactic definition of type theories (SOGATs with two fixed sorts: types and terms) is described by Bauer and Haselwarter [30] based on earlier work [13]. Presentation-independent definitions of SOGATs are representable map categories by Uemura [52] and CwFs with a sort of locally representable types (CwF^+) [15]. The presentation-independent ways define models using functorial semantics, while the ToS approach defines semantics of GATs by induction on the signature. Functorial semantics for our SOGAT signatures is as follows: every SOGAT signature Ω gives rise to the free CwF^+ over Ω (the slice of the theory of SOGAT signatures over Ω). Now a model is a category $\mathcal{C}$ together with a CwF^+-morphism from this CwF^+ to the CwF^+ of presheaves over $\mathcal{C}$.

Our two different ways of translating SOGATs to GATs roughly correspond to Voevodsky's two different descriptions of the substitution calculus for dependent type theory: B-systems correspond to single substitutions, C-systems to parallel substitutions. B-systems and C-systems are equivalent [1], however our single substitution calculus is more minimalistic, and has more models than the parallel substitution calculus.

In this paper we explain how to define languages as SOGATs and then translate them into GATs. Then, the induction principle of the GAT can be used to prove properties of the syntax. However, certain metatheoretic proofs can be described at the level of SOGATs avoiding mentioning contexts or substitutions. Synthethic Tait computability [50] and internal sconing [16] are techniques for this. We leave adapting them to ToS$^+$ as future work.

Metatheory and notation. Our metatheory is extensional type theory with uniqueness of identity proofs, we use Agda-like notation with implicit arguments sometimes omitted. We write function application as juxtaposition, the universe of types is denoted Set_i, we usually omit the level subscripts. We use infix Σ type notation using $\times$, the single element of the singleton type $\mathbb{1}$ is denoted $\star$. Sometimes we work in the internal language of a presheaf category using the same notations, in the style of two-level type theory [4, 10].

2 Classes of algebraic theories through examples

In this section, we walk through examples of logic and programming languages defined as algebraic theories: we define a single-sorted algebraic theory (AT), a generalised algebraic theory (GAT), a second-order algebraic theory (SOAT) and multiple second-order generalised algebraic theories (SOGATs). GATs include typing information compared to ATs, SOATs include binders, while SOGATs combine these two aspects.

2.1 Algebraic theories

Combinator calculus is an algebraic theory (AT) with a single sort of terms, one binary, two nullary operations and two equations. We denote its signature as follows (unlike usual presentations of algebraic theories, we include the equations in the notion of signature, because for generalised algebraic theories separation is not possible).

▶ **Definition 1** (Schönfinkel's combinator calculus)**.**

$$\begin{array}{lll} \mathsf{Tm} : \mathsf{Set} & \mathsf{K} : \mathsf{Tm} & \mathsf{K}\beta : \mathsf{K} \cdot u \cdot f = u \\ \_ \cdot \_ : \mathsf{Tm} \to \mathsf{Tm} \to \mathsf{Tm} & \mathsf{S} : \mathsf{Tm} & \mathsf{S}\beta : \mathsf{S} \cdot f \cdot g \cdot u = f \cdot u \cdot (g \cdot u) \end{array}$$

The notion of algebra/model is evident from this signature. The quotiented syntax of combinator calculus is the initial model, which always exists. Notions of homomorphism, displayed/dependent model, induction, products and coproducts of models, free models, and so on, are derivable from the signature, as described in any book on universal algebra. The initial algebra of an AT is called a quotient inductive type [22].

Single-sorted algebraic theories from logic are classical (or intuitionistic) propositional logic defined as the theory of Boolean algebras (or Heyting algebras). Examples from algebra are monoids, groups, rings, lattices, and so on.

2.2 Generalised algebraic theories

Generalised algebraic theories (GATs) allow sorts indexed by other sorts. Examples are typed combinator calculus and propositional logic with Hilbert-style proof theory, theories of graphs, preorders, categories, and so on.

▶ **Definition 2** (Typed combinator calculus)**.**

$$\begin{array}{lll}
\mathsf{Ty} & : \mathsf{Set} & \mathsf{K} \;\; : \mathsf{Tm}\,(A \Rightarrow B \Rightarrow A) \\
\mathsf{Tm} & : \mathsf{Ty} \to \mathsf{Set} & \mathsf{S} \;\; : \mathsf{Tm}\,\big((A \Rightarrow B \Rightarrow C) \Rightarrow (A \Rightarrow B) \Rightarrow A \Rightarrow C\big) \\
\iota & : \mathsf{Ty} & \mathsf{K}\beta : \mathsf{K} \cdot u \cdot f = u \\
- \Rightarrow - & : \mathsf{Ty} \to \mathsf{Ty} \to \mathsf{Ty} & \mathsf{S}\beta : \mathsf{S} \cdot f \cdot g \cdot u = f \cdot u \cdot (g \cdot u) \\
- \cdot - & : \mathsf{Tm}\,(A \Rightarrow B) \to \mathsf{Tm}\,A \to \mathsf{Tm}\,B &
\end{array}$$

We have a sort of types, and for each type, a separate sort of terms of that type. Now the K and S operations are nullary only in the sense that they don't take Tm arguments, but they still take two and three Ty arguments, respectively. For readability, these are given implicitly. Similarly, application $- \cdot -$ takes the arguments A and B implicitly.

The above mentioned universal algebraic features of ATs generalise to GATs [43]. In particular, each GAT has a syntax given by a quotient inductive-inductive type [39], we have free models [43] and cofree models [45].

If the language has variables or binders, we will define it as a second-order theory.

2.3 Second-order algebraic theories

The SOAT of lambda calculus is the following.

▶ **Definition 3** (Lambda calculus)**.**

$$\mathsf{Tm} : \mathsf{Set} \qquad \mathsf{lam} : (\mathsf{Tm} \to \mathsf{Tm}) \to \mathsf{Tm} \qquad - \cdot - : \mathsf{Tm} \to \mathsf{Tm} \to \mathsf{Tm} \qquad \beta : \mathsf{lam}\, f \cdot u = f\, u$$

The type of lam is not first-order (not strictly positive), hence this is not an algebraic theory anymore. It is clear what a second-order model is (a set with a binary operation and a second-order function with the type of lam satisfying the equation β). However, we do not have a usable notion of homomorphism between second-order models M and N: this would be a function $\alpha : \mathsf{Tm}_M \to \mathsf{Tm}_N$ such that $\alpha\,(t \cdot_M u) = \alpha\, t \cdot_N \alpha\, u$ and $\alpha\,(\mathsf{lam}_M\, f) = \mathsf{lam}_N\,(\alpha \circ f \circ ?)$, but we don't know what to put in place of the ?. To talk about homomorphisms or the syntax, we translate the SOAT to a first-order GAT: we add contexts, substitutions, index Tm and all operations by contexts and then lam becomes a first order function taking a term in an extended context as input. The resulting GAT is the following.

▶ **Definition 4** (Lambda calculus as a first-order GAT)**.**

$$
\begin{array}{ll}
\mathsf{Con} & : \mathsf{Set} \\
\mathsf{Sub} & : \mathsf{Con} \to \mathsf{Con} \to \mathsf{Set} \\
- \circ - & : \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Sub}\,\Theta\,\Delta \to \mathsf{Sub}\,\Theta\,\Gamma \\
\mathsf{ass} & : (\gamma \circ \delta) \circ \theta = \gamma \circ (\delta \circ \theta) \\
\mathsf{id} & : \mathsf{Sub}\,\Gamma\,\Gamma \\
\mathsf{idl} & : \mathsf{id} \circ \gamma = \gamma \\
\mathsf{idr} & : \gamma \circ \mathsf{id} = \gamma \\
\diamond & : \mathsf{Con} \\
\epsilon & : \mathsf{Sub}\,\Gamma\,\diamond \\
\diamond\eta & : (\sigma : \mathsf{Sub}\,\Gamma\,\diamond) \to \sigma = \epsilon \\
\mathsf{Tm} & : \mathsf{Con} \to \mathsf{Set} \\
-[-] & : \mathsf{Tm}\,\Gamma \to \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Tm}\,\Delta \\
[\circ] & : t[\gamma \circ \delta] = t[\gamma][\delta]
\end{array}
\qquad
\begin{array}{ll}
[\mathsf{id}] & : t[\mathsf{id}] = t \\
- \triangleright & : \mathsf{Con} \to \mathsf{Con} \\
-, - & : \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Tm}\,\Delta \to \mathsf{Sub}\,\Delta\,(\Gamma \triangleright) \\
\mathsf{p} & : \mathsf{Sub}\,(\Gamma \triangleright)\,\Gamma \\
\mathsf{q} & : \mathsf{Tm}\,(\Gamma \triangleright) \\
\triangleright\beta_1 & : \mathsf{p} \circ (\gamma, t) = \gamma \\
\triangleright\beta_2 & : \mathsf{q}[\gamma, t] = t \\
\triangleright\eta & : \sigma = (\mathsf{p} \circ \sigma, \mathsf{q}[\sigma]) \\
\mathsf{lam} & : \mathsf{Tm}\,(\Gamma \triangleright) \to \mathsf{Tm}\,\Gamma \\
\mathsf{lam}[] & : (\mathsf{lam}\,t)[\gamma] = \mathsf{lam}\,(t[\gamma \circ \mathsf{p}, \mathsf{q}]) \\
- \cdot - & : \mathsf{Tm}\,\Gamma \to \mathsf{Tm}\,\Gamma \to \mathsf{Tm}\,\Gamma \\
\cdot[] & : (t \cdot u)[\gamma] = t[\gamma] \cdot (u[\gamma]) \\
\beta & : \mathsf{lam}\,t \cdot u = t[\mathsf{id}, u]
\end{array}
$$

We explain in more detail how we obtained the GAT of Definition 4 from the SOAT of Definition 3: the GAT starts with a category with a terminal object ($\mathsf{Con}, \ldots, \diamond\eta$), then there is a sort Tm which is now indexed by Con and comes with an instantiation operation $-[-]$ which is functorial ($[\circ]$, $[\mathsf{id}]$). There is a context extension $-\triangleright$ which makes contexts a natural number algebra (with zero $\diamond$ and successor $-\triangleright$). Substitutions are lists of terms, this is expressed by the components $-, -, \ldots, \triangleright\eta$, which can be grouped together into an isomorphism $\mathsf{p} \circ -, \mathsf{q}[-] : \mathsf{Sub}\,\Delta\,(\Gamma \triangleright) \cong \mathsf{Sub}\,\Delta\,\Gamma \times \mathsf{Tm}\,\Delta : -, -$. Now variables are definable as De Bruijn indices: $0 = \mathsf{q}$, $1 = \mathsf{q}[\mathsf{p}]$, $2 = \mathsf{q}[\mathsf{p}][\mathsf{p}]$, and so on. The operations lam and $- \cdot -$ are also (implicitly) indexed by contexts and come equipped with substitution laws ($\mathsf{lam}[]$ and $\cdot[]$). The function in the input of the SOAT presentation of lam becomes a Tm in an extended context. In $\mathsf{lam}[]$, the substitution $(\gamma \circ \mathsf{p}, \mathsf{q}) : \mathsf{Sub}\,(\Delta \triangleright)\,(\Gamma \triangleright)$ is the lifting of $\gamma : \mathsf{Sub}\,\Delta\,\Gamma$ which does not touch the last variable bound by lam. Finally, the metatheoretic function application on the right hand side of the β law in the SOAT presentation becomes an instantiation of the last variable by $(\mathsf{id}, u) : \mathsf{Sub}\,\Gamma\,(\Gamma \triangleright)$.

In the special case of the lambda calculus, there are equivalent simpler GATs, but this is the one which is generated by the translation of Section 6. Our translation will work generically for any SO(G)AT, hence it does not necessarily give the most minimal GAT presentation.

By the syntax of lambda calculus, we mean the syntax for the GAT of Definition 4. However, we still prefer to define lambda calculus as a SOGAT: it is a shorter definition, does not include boilerplate, and ensures that once translated to its first-order version, all operations respect substitution by construction. Also, we can do programming using the second-order representation in the style of logical frameworks. This means that using the second-order presentation, we can define *derivable* operations and prove derivable equations as opposed to *admissible* ones for which we would need induction. An example of a derivable operation is the Y combinator: we assume a second-order model of the lambda calculus given by Tm, lam, $- \cdot -$, β, and define $\mathsf{Y} := \mathsf{lam}\,\lambda f.\big(\mathsf{lam}\,\lambda x.f \cdot (x \cdot x)\big) \cdot \big(\mathsf{lam}\,\lambda x.f \cdot (x \cdot x)\big)$. We prove that this is indeed a fixpoint combinator as follows.

$$\begin{aligned}
\mathsf{Y}\cdot f = \Big(\mathsf{lam}\,\lambda f.\big(\mathsf{lam}\,\lambda x.f\cdot(x\cdot x)\big)\cdot\big(\mathsf{lam}\,\lambda x.f\cdot(x\cdot x)\big)\Big)\cdot f &= (\beta)\\
\big(\mathsf{lam}\,\lambda x.f\cdot(x\cdot x)\big)\cdot\big(\mathsf{lam}\,\lambda x.f\cdot(x\cdot x)\big) &= (\beta)\\
f\cdot\Big(\big(\mathsf{lam}\,\lambda x.f\cdot(x\cdot x)\big)\cdot\big(\mathsf{lam}\,\lambda x.f\cdot(x\cdot x)\big)\Big) &= f\cdot(\mathsf{Y}\cdot f)
\end{aligned}$$

This kind of reasoning makes sense for any second-order model, and any first-order model gives rise to a second-order model in the internal language of presheaves over the first-order model, see Corollary 28.

2.4 Second-order generalised algebraic theories

SOGATs combine the two previous classes: sorts can be indexed over previous sorts and second-order operations are allowed. In the following examples, we write $\mathsf{f} : \mathsf{A} \leftrightarrow \mathsf{B} : \mathsf{g}$ for $\mathsf{f} : \mathsf{A} \to \mathsf{B}$ and $\mathsf{g} : \mathsf{B} \to \mathsf{A}$, we write $\mathsf{f} : \mathsf{A} \cong \mathsf{B} : \mathsf{g}$ for $\mathsf{f} : \mathsf{A} \leftrightarrow \mathsf{B} : \mathsf{g}$ with two equations $\beta : \mathsf{g}\,(\mathsf{f}\,a) = a$ and $\eta : \mathsf{f}\,(\mathsf{g}\,b) = b$. We write $\mathsf{A} : \mathsf{Prop}$ for $\mathsf{A} : \mathsf{Set}$ together with an equation $\mathsf{irr} : (a\,a' : \mathsf{A}) \to a = a'$. We list the theories as SOGATs, and discuss the interesting aspects of their first-order models.

▶ **Definition 5** (Simply typed lambda calculus)**.**

$$\begin{array}{ll}
\mathsf{Ty} \quad : \mathsf{Set} & \mathsf{Tm} : \mathsf{Ty} \to \mathsf{Set}\\
- \Rightarrow - : \mathsf{Ty} \to \mathsf{Ty} \to \mathsf{Ty} & \mathsf{lam} : (\mathsf{Tm}\,A \to \mathsf{Tm}\,B) \cong \mathsf{Tm}\,(A \Rightarrow B) : - \cdot -
\end{array}$$

An alternative popular description of simply typed lambda calculus is when we omit Ty and Tm, write a horizontal line or $\vdash$ for function space, give names to every input of a function (i.e. we write $(a : \mathsf{Tm}\,A) \to \mathsf{Tm}\,B$ instead of $\mathsf{Tm}\,A \to \mathsf{Tm}\,B$) and use named function application written using square brackets (i.e. we write $t[x \mapsto a]$ instead of $t\,a$, where $t : (x : A) \to B[x \mapsto a]$, where $B : (x : A) \to \mathsf{Set}$). Note that there are no rules for typing variables as they are handled by the metatheory.

$$\frac{A \qquad B}{A \Rightarrow B} \qquad \frac{x : A \vdash b : B}{\mathsf{lam}\,x.b : A \Rightarrow B} \qquad \frac{f : A \Rightarrow B \qquad a : A}{f\cdot a : B} \qquad \frac{}{(\mathsf{lam}\,x.b)\cdot a = b[x \mapsto a]} \qquad \frac{f : A \Rightarrow B}{f = \mathsf{lam}\,x.f\cdot x}$$

A first-order model of the simply typed lambda calculus contains a category with a terminal object (Con, Sub and the empty context $\diamond$), two sorts Ty and Tm which are both indexed by contexts, and there are context extension operations both for types and terms (we omit the types of some operations and equations which are the same as in Definition 4):

▶ **Definition 6** (Simply typed lambda calculus as a GAT with both type and term variables)**.**

$$
\begin{array}{ll}
\mathsf{Con}, \mathsf{Sub}, -\circ-, \mathsf{id}, \mathsf{ass}, \mathsf{idl}, \mathsf{idr}, \diamond, \epsilon, \diamond\eta & \\
\mathsf{Ty} & : \mathsf{Con} \to \mathsf{Set} \\
-[-]_{\mathsf{Ty}} & : \mathsf{Ty}\,\Gamma \to \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Ty}\,\Delta \\
[\circ]_{\mathsf{Ty}}, [\mathsf{id}]_{\mathsf{Ty}} & \\
-\triangleright_{\mathsf{Ty}} & : \mathsf{Con} \to \mathsf{Con} \\
\mathsf{p}_{\mathsf{Ty}} \circ -, \mathsf{q}_{\mathsf{Ty}}[-] & : \mathsf{Sub}\,\Delta\,(\Gamma \triangleright_{\mathsf{Ty}}) \cong \mathsf{Sub}\,\Delta\,\Gamma \times \mathsf{Ty}\,\Delta : -,_{\mathsf{Ty}} - \\
\mathsf{Tm} & : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Set} \\
-[-]_{\mathsf{Tm}} & : \mathsf{Tm}\,\Gamma\,A \to (\gamma : \mathsf{Sub}\,\Delta\,\Gamma) \to \mathsf{Tm}\,\Delta\,(A[\gamma]_{\mathsf{Ty}}) \\
[\circ]_{\mathsf{Tm}}, [\mathsf{id}]_{\mathsf{Tm}} & \\
-\triangleright_{\mathsf{Tm}} - & : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Con} \\
\mathsf{p}_{\mathsf{Tm}} \circ -, \mathsf{q}_{\mathsf{Tm}}[-] & : \mathsf{Sub}\,\Delta\,(\Gamma \triangleright_{\mathsf{Tm}} A) \cong (\gamma : \mathsf{Sub}\,\Delta\,\Gamma) \times \mathsf{Tm}\,\Delta\,(A[\gamma]_{\mathsf{Ty}}) : -,_{\mathsf{Tm}} - \\
- \Rightarrow - & : \mathsf{Ty}\,\Gamma \to \mathsf{Ty}\,\Gamma \to \mathsf{Ty}\,\Gamma \\
\Rightarrow[] & : (A \Rightarrow B)[\gamma] = (A[\gamma]) \Rightarrow (B[\gamma]) \\
\mathsf{lam} & : \mathsf{Tm}\,(\Gamma \triangleright_{\mathsf{Tm}} A)\,(B[\mathsf{p}_{\mathsf{Tm}}]) \to \mathsf{Tm}\,\Gamma\,(A \Rightarrow B) \\
\mathsf{lam}[] & : (\mathsf{lam}\,t)[\gamma] = \mathsf{lam}\,(t[\gamma \circ \mathsf{p}_{\mathsf{Tm}},_{\mathsf{Tm}} \mathsf{q}_{\mathsf{Tm}}]) \\
- \cdot - & \mathsf{Tm}\,\Gamma\,(A \Rightarrow B) \to \mathsf{Tm}\,\Gamma\,A \to \mathsf{Tm}\,\Gamma\,B \\
\cdot[] & : (t \cdot u)[\gamma] = t[\gamma] \cdot (u[\gamma]) \\
\Rightarrow\beta & : \mathsf{lam}\,t \cdot u = t[\mathsf{id}, u] \\
\Rightarrow\eta & : t = \mathsf{lam}\,(t[\mathsf{p}_{\mathsf{Tm}}] \cdot \mathsf{q}_{\mathsf{Tm}})
\end{array}
$$

The context extension operations take as arguments the index of the corresponding sort: Ty is not indexed, so $\triangleright_{\mathsf{Ty}}$ does not take any arguments, $\triangleright_{\mathsf{Tm}}$ takes a Ty argument. In simply typed lambda calculus, none of the operations (or sorts) use type variables, hence it is not necessary to include the operation $\triangleright_{\mathsf{Ty}}$ and the type variables q_{Ty}, $\mathsf{q}_{\mathsf{Ty}}[\mathsf{p}]$, $\mathsf{q}_{\mathsf{Ty}}[\mathsf{p}][\mathsf{p}]$, and so on. In the formal version of signatures (Definition 13), we will distinguish those sorts which have variables and those which do not, so this optimisation can be handled by our setup. The fact that all types are closed (don't depend on term variables, hence do not depend on the context at all) will not be handled by our translation, so the generated theory will include unnecessary dependencies, and a by hand optimisation step is needed to replace $\mathsf{Ty} : \mathsf{Con} \to \mathsf{Set}$ by $\mathsf{Ty} : \mathsf{Set}$ and removing the $-[-]_{\mathsf{Ty}}$ operation. The operations in the notion of first-order model are the typed versions of the operations in Definition 4. Lambda and application could have been presented by an isomorphism $\mathsf{lam} : \mathsf{Tm}\,(\Gamma \triangleright_{\mathsf{Tm}} A)\,(B[\mathsf{p}_{\mathsf{Tm}}]) \cong \mathsf{Tm}\,\Gamma\,(A \Rightarrow B) : \mathsf{app}$, using a unary app application operation instead of the binary $-\cdot-$. Our setup allows choosing between the two versions, see the discussion after Definition 13. This concludes the typed lambda calculus example.

The following definition of first-order logic has minimal amount of logical connectives, but illustrates the general idea. The proof theory that comes with it is natural deduction style, it can be also written following the above conventions using horizontal lines and $\vdash$.

▶ **Definition 7** (Minimal intuitionistic first-order logic)**.**

$$
\begin{array}{ll}
\mathsf{For} & : \mathsf{Set} \\
\mathsf{Tm} & : \mathsf{Set} \\
- \supset - & : \mathsf{For} \to \mathsf{For} \to \mathsf{For} \\
\forall & : (\mathsf{Tm} \to \mathsf{For}) \to \mathsf{For} \\
\mathsf{Eq} & : \mathsf{Tm} \to \mathsf{Tm} \to \mathsf{For}
\end{array}
\qquad
\begin{array}{ll}
\mathsf{Pf} & : \mathsf{For} \to \mathsf{Prop} \\
\mathsf{intro}^{\supset} & : (\mathsf{Pf}\, A \to \mathsf{Pf}\, B) \leftrightarrow \mathsf{Pf}\, (A \supset B) : \mathsf{elim}^{\supset} \\
\mathsf{intro}^{\forall} & : \big((t : \mathsf{Tm}) \to \mathsf{Pf}\, (A\, t)\big) \leftrightarrow \mathsf{Pf}\, (\forall A) : \mathsf{elim}^{\forall} \\
\mathsf{intro}^{\mathsf{Eq}} & : \mathsf{Pf}\, (\mathsf{Eq}\, t\, t) \\
\mathsf{elim}^{\mathsf{Eq}} & : (A : \mathsf{Tm} \to \mathsf{For}) \to \mathsf{Pf}\, (\mathsf{Eq}\, t\, t') \to \\
 & \quad \mathsf{Pf}\, (A\, t) \to \mathsf{Pf}\, (A\, t')
\end{array}
$$

A first-order model contains a category of contexts and substitutions equipped with three different kinds of context extension corresponding to three different kinds of variables. This means that there are three different 0 De Bruijn indices ($\mathsf{q}_{\mathsf{For}}$, q_{Tm}, q_{Pf}), nine different 1 De Bruijn indices ($\mathsf{q}_{\mathsf{For}}[\mathsf{p}_{\mathsf{For}}]_{\mathsf{For}}$, $\mathsf{q}_{\mathsf{For}}[\mathsf{p}_{\mathsf{Tm}}]_{\mathsf{For}}$, $\mathsf{q}_{\mathsf{For}}[\mathsf{p}_{\mathsf{Pf}}]_{\mathsf{For}}$, $\dots$, $\mathsf{q}_{\mathsf{Pf}}[\mathsf{p}_{\mathsf{Pf}}]_{\mathsf{Pf}}$). In general, De Bruijn index n has 3^{n+1} variants. We list the types of the binders:

$$
\begin{array}{ll}
\forall & : \mathsf{For}\, (\Gamma \triangleright_{\mathsf{Tm}}) \to \mathsf{For}\, \Gamma \\
\mathsf{intro}^{\supset} & : \mathsf{Pf}\, (\Gamma \triangleright_{\mathsf{Pf}} A)\, (B[\mathsf{p}_{\mathsf{Pf}}]_{\mathsf{For}}) \to \mathsf{Pf}\, \Gamma\, (A \supset B) \\
\mathsf{intro}^{\forall} & : \mathsf{Pf}\, (\Gamma \triangleright_{\mathsf{Tm}})\, A \to \mathsf{Pf}\, \Gamma\, (\forall A) \\
\mathsf{elim}^{\mathsf{Eq}} & : \big(A : \mathsf{For}\, (\Gamma \triangleright_{\mathsf{Tm}})\big) \to \mathsf{Pf}\, \Gamma\, (\mathsf{Eq}\, t\, t') \to \mathsf{Pf}\, \Gamma\, (A[\mathsf{id}\, ,_{\mathsf{Tm}}\, t]_{\mathsf{For}}) \to \mathsf{Pf}\, \Gamma\, (A[\mathsf{id}\, ,_{\mathsf{Tm}}\, t']_{\mathsf{For}})
\end{array}
$$

The GAT presentation of first-order logic can be simplified by removing For variables as no operations bind formulas. Another post-hoc simplification is separating the Tm-variable contexts and the Pf-variable contexts which depend on the former. After such a separation, it is possible to define [11] the syntax of first-order logic simply using inductive types and avoiding quotienting (with the exception of Pf where we use a full quotient which can be implemented by SProp of Agda or Coq [25]). One reason for being able to do this is that the above SOGAT does not have any equations, but this is not enough in general. For example, if we do not have quotients, it does not seem to be possible to define the syntax of a Martin-Löf type theory without computation rules.

Next we show the SOGAT definition of the polymorphic lambda calculus.

▶ **Definition 8** (System F)**.**

$$
\begin{array}{ll}
\mathsf{Ty} & : \mathsf{Set} \\
\mathsf{Tm} & : \mathsf{Ty} \to \mathsf{Set} \\
- \Rightarrow - & : \mathsf{Ty} \to \mathsf{Ty} \to \mathsf{Ty} \\
\mathsf{lam} & : (\mathsf{Tm}\, A \to \mathsf{Tm}\, B) \cong \mathsf{Tm}\, (A \Rightarrow B) : - \cdot - \\
\forall & : (\mathsf{Ty} \to \mathsf{Ty}) \to \mathsf{Ty} \\
\mathsf{Lam} & : \big((X : \mathsf{Ty}) \to \mathsf{Tm}\, (A\, X)\big) \cong \mathsf{Tm}\, (\forall\, A) : - \bullet -
\end{array}
$$

The first order version is Definition 6 extended with the following operations and equations for $\forall$. Now we really need both type and term variables. We use a unary application operation for $\forall$, see discussion after Definition 13.

$$
\begin{array}{ll}
\forall & : \mathsf{Ty}\, (\Gamma \triangleright_{\mathsf{Ty}}) \to \mathsf{Ty}\, \Gamma \\
\forall[] & : (\forall\, A)[\gamma] = \forall\, (A[\gamma \circ \mathsf{p}_{\mathsf{Ty}} \triangleright_{\mathsf{Ty}} \mathsf{q}_{\mathsf{Ty}}])
\end{array}
\qquad
\begin{array}{ll}
\mathsf{Lam} & : \mathsf{Tm}\, (\Gamma \triangleright_{\mathsf{Ty}})\, A \cong \mathsf{Tm}\, \Gamma\, (\forall\, A) : \mathsf{App} \\
\mathsf{Lam}[] & : (\mathsf{Lam}\, t)[\gamma] = \mathsf{Lam}\, (t[\gamma \circ \mathsf{p}_{\mathsf{Ty}} \triangleright_{\mathsf{Ty}} \mathsf{q}_{\mathsf{Ty}}])
\end{array}
$$

The next language is interesting because its sorts and operations are interleaved: the typing of the sort Tm depends on the operation $*$.

▶ **Definition 9** (System F_ω).

$$
\begin{array}{ll}
\Box & : \mathsf{Set} \\
\mathsf{Ty} & : \Box \to \mathsf{Set} \\
- \Rrightarrow - & : \Box \to \Box \to \Box \\
\mathsf{LAM} & : (\mathsf{Ty}\, K \to \mathsf{Ty}\, L) \cong \\
 & \quad \mathsf{Ty}\,(K \Rrightarrow L) : - \bullet - \\
* & : \Box
\end{array}
\qquad
\begin{array}{ll}
\mathsf{Tm} & : \mathsf{Ty}\, * \to \mathsf{Set} \\
\forall & : (\mathsf{Ty}\, K \to \mathsf{Ty}\, *) \to \mathsf{Ty}\, * \\
\mathsf{Lam} & : ((X : \mathsf{Ty}\, K) \to \mathsf{Tm}\,(A\, X)) \cong \\
 & \quad \mathsf{Tm}\,(\forall\, A) : - \bullet - \\
- \Rightarrow - & : \mathsf{Ty}\, * \to \mathsf{Ty}\, * \to \mathsf{Ty}\, * \\
\mathsf{lam} & : (\mathsf{Tm}\, A \to \mathsf{Tm}\, B) \cong \mathsf{Tm}\,(A \Rightarrow B) : - \cdot -
\end{array}
$$

In the first-order version (minimised by removing $\Box$ (kind) variables), we have sorts $\Box : \mathsf{Set}$, $\mathsf{Ty} : \mathsf{Con} \to \Box \to \mathsf{Set}$, an operation $* : \Box$, and a sort $\mathsf{Tm} : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma\, * \to \mathsf{Set}$. We have three operations binding Ty-variables and one operation binding a term-variable:

$$
\begin{array}{ll}
\mathsf{LAM} : \mathsf{Ty}\,(\Gamma \triangleright_{\mathsf{Ty}} K)\, L \to \mathsf{Ty}\,\Gamma\,(K \Rrightarrow L) & \mathsf{Lam} : \mathsf{Tm}\,(\Gamma \triangleright_{\mathsf{Ty}} K)\, A \to \mathsf{Tm}\,\Gamma\,(\forall A) \\
\forall \quad : \mathsf{Ty}\,(\Gamma \triangleright_{\mathsf{Ty}} K) * \to \mathsf{Ty}\,\Gamma\, * & \mathsf{lam} \ : \mathsf{Tm}\,(\Gamma \triangleright_{\mathsf{Tm}} A)\,(B[\mathsf{p}_{\mathsf{Tm}}]_{\mathsf{Ty}}) \to \mathsf{Tm}\,\Gamma\,(A \Rightarrow B)
\end{array}
$$

Our next example is a theory with dependent types featuring Π types, a Coquand-universe (which forces types to be indexed by levels) and a lifting operation. This is an open signature which means that it refers to some external types, in this case a natural number algebra (we can make it closed by adding $\mathbb{N}$ as a new sort and 0 and $1 + -$ as new operations).

▶ **Definition 10** (Minimal Martin-Löf type theory).

$$
\begin{array}{ll}
\mathsf{Ty} & : \mathbb{N} \to \mathsf{Set} \\
\mathsf{Tm} & : \mathsf{Ty}\, i \to \mathsf{Set} \\
\Pi & : (A : \mathsf{Ty}\, i) \to (\mathsf{Tm}\, A \to \mathsf{Ty}\, i) \to \mathsf{Ty}\, i \\
\mathsf{lam} & : ((a : \mathsf{Tm}\, A) \to \mathsf{Tm}\,(B\, a)) \cong \mathsf{Tm}\,(\Pi\, A\, B) : - \cdot -
\end{array}
\qquad
\begin{array}{ll}
\mathsf{U} & : (i : \mathbb{N}) \to \mathsf{Ty}\,(1 + i) \\
\mathsf{c} & : \mathsf{Ty}\, i \cong \mathsf{Tm}\,(\mathsf{U}\, i) : \mathsf{El} \\
\mathsf{Lift} & : \mathsf{Ty}\, i \to \mathsf{Ty}\,(1 + i) \\
\mathsf{mk} & : \mathsf{Tm}\, A \cong \mathsf{Tm}\,(\mathsf{Lift}\, A) : \mathsf{un}
\end{array}
$$

The first-order translation of this theory results in a category with families (CwF [19]), more precisely, a category with $\mathbb{N}$-many families equipped with familywise Π-types, universes and a one-step upwards lifting between the families. The sorts are $\mathsf{Ty} : \mathsf{Con} \to \mathbb{N} \to \mathsf{Set}$ and $\mathsf{Tm} : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma\, i \to \mathsf{Set}$, the i argument is implicit in the latter.

Instead of a Coquand-universe with c and El, we could have defined a Russell universe where we have a sort equality $\mathsf{Ty}\, i = \mathsf{Tm}\,(\mathsf{U}\, i)$, and we also have the option to do this for lifting and Π types. The first-order semantics of such a theory has the following equalities where the second one makes sense because of the first one: $\mathsf{Ty}\,\Gamma\, i = \mathsf{Tm}\,\Gamma\,(\mathsf{U}\, i)$, $A[\gamma]_{\mathsf{Ty}} = A[\gamma]_{\mathsf{Tm}}$. Having strict Π types means $\mathsf{Tm}\,(\Gamma \triangleright A)\, B = \mathsf{Tm}\,\Gamma\,(\Pi\, A\, B)$ and $t[\gamma] = t[\gamma \circ \mathsf{p}, \mathsf{q}]$ where the left hand side t is in $\mathsf{Tm}\,\Gamma\,(\Pi\, A\, B)$.

3 Theories of signatures as SOGATs

In this section we define three languages which describe signatures for ATs, GATs and SOGATs, respectively. All three languages are given as SOGATs.

The theory of signatures for ATs is a dependent type theory without a universe, it has one base type Srt for the (single) sort, Σ types, a Π type with fixed Srt domain, and an equality type. Π types are equipped with application, but the Σ and Eq types don't have constructors or destructors, because those are not needed when defining signatures.

▶ **Definition 11** (Signatures for single-sorted algebraic theories)**.**

$$
\begin{array}{ll}
\mathsf{Ty} & : \mathsf{Set} \\
\mathsf{Tm} & : \mathsf{Ty} \to \mathsf{Set} \\
\Sigma & : (A : \mathsf{Ty}) \to (\mathsf{Tm}\, A \to \mathsf{Ty}) \to \mathsf{Ty} \\
\mathsf{Srt} & : \mathsf{Ty}
\end{array}
\qquad
\begin{array}{ll}
\Pi\mathsf{Srt} & : (\mathsf{Tm}\, \mathsf{Srt} \to \mathsf{Ty}) \to \mathsf{Ty} \\
- \cdot - & : \mathsf{Tm}\, (\Pi\mathsf{Srt}\, B) \to (x : \mathsf{Tm}\, \mathsf{Srt}) \to \mathsf{Tm}\, (B\, x) \\
\mathsf{Eq} & : \mathsf{Tm}\, \mathsf{Srt} \to \mathsf{Tm}\, \mathsf{Srt} \to \mathsf{Ty}
\end{array}
$$

A first-order model of this theory is a CwF with type formers Σ, Srt, $\Pi\mathsf{Srt}$, Eq and a term former $- \cdot - : \mathsf{Tm}\, \Gamma\, (\Pi\mathsf{Srt}\, B) \to (x : \mathsf{Tm}\, \Gamma\, \mathsf{Srt}) \to \mathsf{Tm}\, \Gamma\, (B[\mathsf{id}, x])$. An element of Ty in the syntax of this language is an AT signature. We introduce abbreviations $\mathsf{Srt} \Rightarrow A := \Pi\mathsf{Srt}\, \lambda\_.A$ and $A \times B := \Sigma\, A\, \lambda\_.B$. The signature for combinator calculus is the following Ty:

$$
\begin{aligned}
&\Sigma\, (\mathsf{Srt} \Rightarrow \mathsf{Srt} \Rightarrow \mathsf{Srt})\, \lambda app.\Sigma\, \mathsf{Srt}\, \lambda K.\Sigma\, \mathsf{Srt}\, \lambda S.\Big(\Pi\mathsf{Srt}\, \lambda u.\Pi\mathsf{Srt}\, \lambda f.\mathsf{Eq}\, \big(app \cdot (app \cdot K \cdot u) \cdot f\big)\, u\Big) \\
&\times \Big(\Pi\mathsf{Srt}\, \lambda f.\Pi\mathsf{Srt}\, \lambda g.\Pi\mathsf{Srt}\, \lambda u.\mathsf{Eq}\, \Big(app \cdot \big(app \cdot (app \cdot S \cdot f) \cdot g\big) \cdot u\Big) \\
&\qquad\qquad \Big(app \cdot (app \cdot f \cdot u) \cdot (app \cdot g \cdot u)\Big)\Big)
\end{aligned}
$$

This can be seen as a more explicit version of Definition 1: we use Σ types instead of a newline-separated list, we use the metatheoretic λ binder to give names to operations, we use an explicit $\cdot$ operation for application and write Eq instead of $=$. Moreover, we don't have infix operators or implicit arguments, the three arguments of equation $\mathsf{K}\beta$ and the four arguments of equation $\mathsf{S}\beta$ have to be introduced using $\Pi\mathsf{Srt}$ explicitly. Being more explicit is needed to make sure that we describe an algebraic theory: for example, the fact that the domain of Π is fixed ensures strict positivity.

The theory of GAT signatures (ToS) is a type theory with an empty universe (a type and a family over it), $\top$ and Σ types, equality with reflection, and a Π type with U-domain.

▶ **Definition 12** (ToS: the theory of GAT signatures)**.**

$$
\begin{array}{ll}
\mathsf{Ty} & : \mathsf{Set} \\
\mathsf{Tm} & : \mathsf{Ty} \to \mathsf{Set} \\
\mathsf{U} & : \mathsf{Ty} \\
\mathsf{El} & : \mathsf{Tm}\, \mathsf{U} \to \mathsf{Ty} \\
\top & : \mathsf{Ty} \\
\mathsf{tt} & : \mathbb{1} \cong \mathsf{Tm}\, \top
\end{array}
\qquad
\begin{array}{ll}
\Sigma & : (A : \mathsf{Ty}) \to (\mathsf{Tm}\, A \to \mathsf{Ty}) \to \mathsf{Ty} \\
(-, -) & : (a : \mathsf{Tm}\, A) \times \mathsf{Tm}\, (B\, a) \cong \mathsf{Tm}\, (\Sigma\, A\, B) : \mathsf{fst}, \mathsf{snd} \\
\Pi & : (a : \mathsf{Tm}\, \mathsf{U}) \to (\mathsf{Tm}\, (\mathsf{El}\, a) \to \mathsf{Ty}) \to \mathsf{Ty} \\
\mathsf{lam} & : \big((x : \mathsf{Tm}\, (\mathsf{El}\, a)) \to \mathsf{Tm}\, (B\, x)\big) \cong \mathsf{Tm}\, (\Pi\, a\, B) : - \cdot - \\
\mathsf{Eq} & : (A : \mathsf{Ty}) \to \mathsf{Tm}\, A \to \mathsf{Tm}\, A \to \mathsf{Ty} \\
\mathsf{refl} & : (u = v) \cong \mathsf{Tm}\, (\mathsf{Eq}\, A\, u\, v) : \mathsf{reflect}
\end{array}
$$

The first-order version is Definition 14. A (presentation of a) GAT is defined as a closed type in the syntax of ToS. The base type U is for declaring sorts, so a signature has to start with a sort, and then we can declare elements of the sort using El or functions where the input is a sort. For example, part of typed combinator calculus (Definition 2) is given by the following signature. We use the abbreviations $a \Rightarrow B := \Pi\, a\, \lambda\_.b$ and $A \times B = \Sigma\, A\, \lambda\_.B$. We left out the S combinator and its β rule for reasons of space.

$$\Sigma\,\mathsf{U}\,\lambda\,Ty.\Sigma\,(\,Ty \Rightarrow \mathsf{U})\,\lambda\,Tm.\mathsf{El}\;Ty \times \Sigma\,(\,Ty \Rightarrow\; Ty \Rightarrow \mathsf{El}\;Ty)\,\lambda\,arr.\Sigma$$
$$\big(\Pi\;Ty\,\lambda A.\Pi\;Ty\,\lambda B.\,Tm\cdot(arr\cdot A\cdot B) \Rightarrow\; Tm\cdot A \Rightarrow \mathsf{El}\,(\,Tm\cdot B)\big)\,\lambda\,app.\Sigma$$
$$\Big(\Pi\;Ty\,\lambda A.\Pi\;Ty\,\lambda B.\mathsf{El}\,\Big(\,Tm\cdot\big(arr\cdot A\cdot(arr\cdot B\cdot A)\big)\Big)\Big)\,\lambda K.\Sigma$$
$$\Big(\Pi\;Ty\,\lambda A.\Pi\;Ty\,\lambda B.\Pi\,(\,Tm\cdot A)\,\lambda u.\Pi\,(\,Tm\cdot B)\,\lambda f.\mathsf{Eq}\,\big(\mathsf{El}\,(\,Tm\cdot A)\big)$$
$$\big(app\cdot(arr\cdot B\cdot A)\cdot B\cdot(app\cdot(arr\cdot A\cdot(arr\cdot B\cdot A))\cdot A\cdot K\cdot u)\cdot f\big)\,u\Big)\times\ldots$$

This type is a very explicit version of Definition 2: we use Σ, explicit application $\cdot$, no infix operators, no implicit arguments, and explicit El turning terms in U into types. We expect that an elaboration algorithm can turn Definition 2 into such an explicit version.

For the theory of SOGAT signatures (ToS$^+$), we add a new universe U^+ of sorts for which variables are allowed: with the help of these we can write second order functions. U^+ is a subuniverse of U (witnessed by el^+) and has a Π type with U^+-domain and U-codomain.

▶ **Definition 13** (ToS$^+$: the theory of SOGAT signatures). *We extend ToS with the following.*

$$\mathsf{U}^+ : \mathsf{Ty} \qquad \pi^+ \quad : (a^+ : \mathsf{Tm}\,\mathsf{U}^+) \to \Big(\mathsf{Tm}\,\big(\mathsf{El}\,(\mathsf{el}^+\,a^+)\big) \to \mathsf{Tm}\,\mathsf{U}\Big) \to \mathsf{Tm}\,\mathsf{U}$$
$$\mathsf{el}^+ : \mathsf{Tm}\,\mathsf{U}^+ \to \mathsf{Tm}\,\mathsf{U} \qquad \mathsf{lam}^+ : \big(x : \mathsf{El}\,(\mathsf{el}^+\,a^+)\big) \to \mathsf{Tm}\,(\mathsf{El}\,(b\,x)) \cong \mathsf{Tm}\,\big(\mathsf{El}\,(\pi^+\,a^+\,b)\big) : \_ \cdot^+ \_$$

The first-order version is Definition 15. A (presentation of a) GAT is defined as a closed type in the syntax of ToS$^+$. The signature for lambda calculus (Definition 3) is the following element of Ty.

$$\Sigma\,\mathsf{U}^+\,\lambda\,Tm.\Sigma\,\big((\,Tm \Rightarrow^+ \mathsf{el}^+\;Tm) \Rightarrow \mathsf{El}\,(\mathsf{el}^+\;Tm)\big)\,\lambda\,lam.\Sigma\,\big(\mathsf{el}^+\;Tm \Rightarrow \mathsf{el}^+\;Tm \Rightarrow \mathsf{El}\,(\mathsf{el}^+\;Tm)\big)\,\lambda\,app.$$
$$\Pi\,(\,Tm \Rightarrow^+ \mathsf{el}^+\;Tm)\,\lambda t.\Pi\,(\mathsf{el}^+\;Tm)\,\lambda u.\mathsf{Eq}\,\big(\mathsf{El}\,(\mathsf{el}^+\;Tm)\big)\,\big(app\cdot(lam\cdot t)\cdot u\big)\,(t\cdot^+ u)$$

We have one sort Tm for which variables are allowed, application app uses ordinary function space $\Rightarrow$ where Tm has to be lifted by el^+ from U^+ to U. Lambda lam is defined as a second-order function where $\Rightarrow^+$ can appear on the left hand side of an $\Rightarrow$. When stating the β equation, note the two different application operators ($\cdot$ vs. $\cdot^+$): $\cdot^+$ is used when giving value to a variable. This becomes clear if we look at the first-order presentation of the β law (last line in Definition 4, we write app instead of $\cdot$ to avoid confusion): $app\,(lam\,t)\,u = t[\mathsf{id}, u]$. So the semantics of $\cdot$ should be simply function application, while the semantics of $\cdot^+$ is instantiation with a substitution. We give another illustration of this difference: in the above signature, the type of app is $\mathsf{el}^+\;Tm \Rightarrow \mathsf{el}^+\;Tm \Rightarrow \mathsf{El}\,(\mathsf{el}^+\;Tm)$, and this is translated to $Tm\,\Gamma \to Tm\,\Gamma \to Tm\,\Gamma$ in the GAT version (see Definition 4). But we could have defined app as having type $\mathsf{El}\,(\,Tm \Rightarrow^+ Tm \Rightarrow^+ Tm)$. In this case the GAT version of app would be in $Tm\,(\Gamma \triangleright\triangleright)$. Both variants are meaningful, and ToS$^+$ allows the user to make a choice if she wants an operation with arguments, or an operation returning in an extended context. Note that both function spaces in the type of lam are forced to be $\Rightarrow^+$ and $\Rightarrow$, respectively.

Analogously, all SOGATs in Sections 2, 3 and Appendix A can be reified into SOGAT signatures (with the exception of Martin-Löf type theory which is an open signature, but we will rectify this in Section 7). This includes ToS$^+$ itself.

4 Naive semantics of SOGAT signatures

In this section, for any SOGAT signature, we define a notion of first-order model. The idea is that a model is a category together with the presheaf interpretation of the signature over that category: the category of presheaves supports a universe, Π types, and so on, so we directly use these when interpreting the type formers of ToS$^+$. We assume basic working knowledge of categories with families (CwFs [19]).

▶ **Definition 14** (First-order model of ToS). *A first-order model of ToS is a CwF (sorts are denoted* Con, Sub, Ty, Tm, *the empty context is* $\diamond$, *the empty substitution is* $\epsilon : \mathsf{Sub}\,\Gamma\,\diamond$, *context extension is* $-\triangleright-$ *:* $(\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Con}$ *with* $\mathsf{p} \circ -, \mathsf{q}[-] : \mathsf{Sub}\,\Delta\,(\Gamma \triangleright A) \cong (\gamma : \mathsf{Sub}\,\Delta\,\Gamma) \times \mathsf{Tm}\,\Delta\,(A[\gamma]) : -\,,\,-$*) equipped with:*

- $\top$ *and* Σ *types given by isomorphisms*
  $\mathsf{tt} : \mathbb{1} \cong \mathsf{Tm}\,\Gamma\,\top$, $(-,-) : (a : \mathsf{Tm}\,\Gamma\,A) \times \mathsf{Tm}\,\Gamma\,(B[\mathsf{id}, a]) \cong \mathsf{Tm}\,\Gamma\,(\Sigma\,A\,B) : \mathsf{fst}, \mathsf{snd}$.
- *A universe given by* $\mathsf{U} : \mathsf{Ty}\,\Gamma$ *and* $\mathsf{El} : \mathsf{Tm}\,\Gamma\,\mathsf{U} \to \mathsf{Ty}\,\Gamma$.
- *A function space with domain in* U, *that is* $\Pi : (a : \mathsf{Tm}\,\Gamma\,\mathsf{U}) \to \mathsf{Ty}\,(\Gamma \triangleright \mathsf{El}\,a) \to \mathsf{Ty}\,\Gamma$, *with an isomorphism* $\mathsf{lam} : \mathsf{Tm}\,(\Gamma \triangleright \mathsf{El}\,a)\,B \cong \mathsf{Tm}\,\Gamma\,(\Pi\,a\,B) : \mathsf{app}$.
- *A strict equality type* Eq *with reflection and uniqueness of identity proofs.*
- *All the operations listed above are natural in* Γ.

▶ **Definition 15** (First-order model of ToS$^+$). *A first-order model of ToS$^+$ is a first-order model of ToS equipped with:*

- *Another universe* $\mathsf{U}^+ : \mathsf{Ty}\,\Gamma$ *that is a subuniverse of* U *i.e.* $\mathsf{el}^+ : \mathsf{Tm}\,\Gamma\,\mathsf{U}^+ \to \mathsf{Tm}\,\Gamma\,\mathsf{U}$.
- U *is closed under functions with* U^+*-domain, i.e.* $\pi^+ : (a^+ : \mathsf{Tm}\,\Gamma\,\mathsf{U}^+) \to \mathsf{Tm}\,(\Gamma \triangleright \mathsf{El}\,(\mathsf{el}^+\,a^+))\,\mathsf{U} \to \mathsf{Tm}\,\Gamma\,\mathsf{U}$ *with* $\mathsf{lam}^+ : \mathsf{Tm}\,(\Gamma \triangleright \mathsf{El}\,(\mathsf{el}^+\,a))\,(\mathsf{El}\,b) \cong \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,(\pi^+\,a^+\,b)) : \mathsf{app}^+$.
- *All the operations listed above are natural in* Γ.

▶ **Problem 16** (PSh). *Presheaves over a category* C *form a CwF equipped with* $\top$, Σ *types, an equality type with reflection,* Π *types and a Coquand-universe* U *with* $\mathsf{c} : \mathsf{Ty}\,\Gamma \cong \mathsf{Tm}\,\Gamma\,\mathsf{U} : \mathsf{El}$. *Unlike in Definition 10, we omit writing universe indices for readibility.*

Construction. We recall the main parts of the construction [31] for fixing notations. $\Gamma : \mathsf{Con}$ is a presheaf, that is a family of sets $\Gamma : C \to \mathsf{Set}$ with reindexing $\gamma_I[f]_\Gamma : \Gamma\,J$ for $\gamma_I : \Gamma\,I$ and $f : C(J, I)$ such that $\gamma_I[f \circ g]_\Gamma = \gamma_I[f]_\Gamma[g]_\Gamma$ and $\gamma_I[\mathsf{id}]_\Gamma = \gamma_I$. A $\sigma : \mathsf{Sub}\,\Delta\,\Gamma$ is a function $\sigma : \Delta\,I \to \Gamma\,I$ such that $(\sigma\,\delta_I)[f]_\Gamma = \sigma\,(\delta_I[f]_\Delta)$. A type $A : \mathsf{Ty}\,\Gamma$ is a dependent presheaf containing a family $A : (I : C) \to \Gamma\,I \to \mathsf{Set}$ with reindexing $a_I[f]_A : A\,J\,(\gamma_I[f]_\Gamma)$ for $a_I : A\,I\,\gamma_I$ and $f : C(J, I)$ satisfying functoriality. Type substitution is $A[\gamma]\,I\,\delta_I := A\,I\,(\gamma\,\delta_I)$. A term $a : \mathsf{Tm}\,\Gamma\,A$ is a function $a : (\gamma_I : \Gamma\,I) \to A\,I\,\gamma_I$ such that $(a\,\gamma_I)[f]_A = a\,(\gamma_I[f]_\Gamma)$. Term substitution is $a[\gamma]\,\delta_I := a\,(\gamma\,\delta_I)$. The empty context is constant unit: $\diamond\,I := \mathbb{1}$. Context extension is pointwise: $(\Gamma \triangleright A)\,I := (\gamma_I : \Gamma\,I) \times A\,I\,\gamma_I$, its universal property is given by projections and pairing for metatheoretic Σ types. $\top$, Σ and Eq are pointwise. We have the functor Yoneda $\mathsf{y} : C \to \mathsf{PSh}(C)$ defined by $\mathsf{y}\,I\,J := C(J, I)$, and we use this to define the universe by $\mathsf{U}\,I\,\gamma_I := \mathsf{Ty}\,(\mathsf{y}\,I)$. We observe that $\gamma_I[-]_\Gamma : \mathsf{Sub}\,(\mathsf{y}\,I)\,\Gamma$ (forward part of Yoneda lemma), and define $\Pi\,A\,B\,I\,\gamma_I := \mathsf{Tm}\,(\mathsf{y}\,I \triangleright A[\gamma_I[-]_\Gamma])\,(B[\gamma_I[-]_\Gamma \circ \mathsf{p}, \mathsf{q}])$. ◀

▶ **Problem 17** (Locally representable types). *The CwF of presheaves can be extended to a CwF$^+$, which means a CwF with a subsort of* Ty *called* Ty^+ *and a* Π^+ *type with domain in* Ty^+, *i.e.* $\Pi^+ : (A : \mathsf{Ty}^+\,\Gamma) \to \mathsf{Ty}\,(\Gamma \triangleright A) \to \mathsf{Ty}\,\Gamma$ *with* $\mathsf{lam}^+ : \mathsf{Tm}\,(\Gamma \triangleright A)\,B \cong \mathsf{Tm}\,\Gamma\,(\Pi^+\,A\,B) : \mathsf{app}^+$, *natural in* Γ. Ty^+ *is classified by the Coquand universe* U^+.

Construction. An element $A : \mathsf{Ty}^+\,\Gamma$ is an $A : \mathsf{Ty}\,\Gamma$ together with $- \triangleright_A - : (I : C) \to \Gamma\,I \to C$ and an isomorphism $\mathsf{p}_A \circ -, \mathsf{q}_A[-]_A : C(J, I \triangleright_A \gamma_I) \cong (f : C(J,I)) \times A\,J\,(\gamma_I[f]_\Gamma) : -,_A -$ natural in J. So $\mathsf{p}_A : C(I \triangleright_A \gamma_I, I)$ and $\mathsf{q}_A : A\,(I \triangleright_A \gamma_I)\,(\gamma_I[\mathsf{p}_A]_\Gamma)$. Substitution is given by $I \triangleright_{A[\gamma]} \delta_I := I \triangleright_A \gamma\,\delta_I$ and we have $C(J, I \triangleright_{A[\gamma]} \delta_I) = C(J, I \triangleright_A \gamma\,\delta_I) \cong (f : C(J,I)) \times A\,J\,(\gamma\,\delta_I[f]_\Gamma) = (f : C(J,I)) \times A[\gamma]\,J\,(\delta_I[f]_\Delta)$. We define Π^+ using the $\triangleright_A$ operator which comes with A, i.e. $\Pi^+\,A\,B\,I\,\gamma_I := B\,(I \triangleright_A \gamma_I)\,(\gamma_I[\mathsf{p}_A]_\Gamma, \mathsf{q}_A)$, $b_{I'}[f]_{\Pi^+\,A\,B} := b_{I'}[f \circ \mathsf{p}_A ,_A \mathsf{q}_A]$, $\mathsf{lam}^+\,b\,\gamma_I := b\,(\gamma_I[\mathsf{p}_A]_\Gamma, \mathsf{q}_A)$ and $\mathsf{app}^+\,t\,(\gamma_I, a_I) := (t\,\gamma_I)[\mathsf{id}_I ,_A a_I]_B$. Like U, $\mathsf{U}^+\,I\,\gamma_I := \mathsf{Ty}^+\,(\mathsf{y}\,I)$. ◀

▶ **Definition 18** (Naive semantics). *Given a category C, $\mathsf{PSh}(C)$ is a model of ToS$^+$ choosing $\mathsf{U} := \mathsf{U}$, $\mathsf{El}\,a := \mathsf{El}\,a$, $\Pi\,a\,B := \Pi\,(\mathsf{El}\,a)\,B$, $\mathsf{U}^+ := \mathsf{U}^+$, $\mathsf{el}^+\,a^+ := \mathsf{c}\,(\mathsf{El}^+\,a^+)$, $\pi^+\,a^+\,b := \mathsf{c}\,(\Pi^+\,(\mathsf{El}^+\,a^+)\,(\mathsf{El}\,b))$. Recall that a SOGAT signature Ω is an element of $\mathsf{Ty}\,\diamond$ in the syntax of ToS$^+$. A naive model of Ω is a category with a terminal object $\diamond$ together with the interpretation of Ω in presheaves over this category, i.e. $(C : \mathsf{Cat}^\diamond) \times \mathsf{Tm}_{\mathsf{PSh}(C)}\,\diamond\,[\![\Omega]\!]_{\mathsf{PSh}(C)}$.*

This definition immediately implies that internally to presheaves over a naive first-order model, we have a second order model.

For illustration, we compute the naive semantics for the signature of untyped lambda calculus without the equations. The informal signature is $\mathsf{Tm} : \mathsf{U}, \mathsf{lam} : (\mathsf{Tm} \to \mathsf{Tm}) \to \mathsf{Tm}, - \cdot - : \mathsf{Tm} \to \mathsf{Tm} \to \mathsf{Tm}$, the second-order formal version is $\Sigma\,\mathsf{U}^+\,\lambda\,Tm.\big((Tm \Rightarrow^+ \mathsf{el}^+\,Tm) \Rightarrow \mathsf{El}\,(\mathsf{el}^+\,Tm)\big) \times (\mathsf{el}^+\,Tm \Rightarrow \mathsf{el}^+\,Tm \Rightarrow \mathsf{El}\,(\mathsf{el}^+\,Tm))$, and we interpret the first-order version of this. We assume a $C : \mathsf{Cat}^\diamond$, write $\mathcal{D} := \mathsf{PSh}(C)$, and use $\mathsf{Tm}_{\mathcal{D}}\,\diamond[\![\Omega]\!]_{\mathcal{D}} \cong [\![\Omega]\!]_{\mathcal{D}}\,\diamond_C\,\star$.

$$
\begin{aligned}
&\Big[\!\Big[\Sigma\,\mathsf{U}^+\,\Big(((\mathsf{q} \Rightarrow^+ \mathsf{el}^+\,\mathsf{q}) \Rightarrow \mathsf{El}\,(\mathsf{el}^+\,\mathsf{q})) \times \big(\mathsf{el}^+\,\mathsf{q} \Rightarrow \mathsf{el}^+\,\mathsf{q} \Rightarrow \mathsf{El}\,(\mathsf{el}^+\,\mathsf{q})\big)\Big)\Big]\!\Big]_{\mathcal{D}}\,\diamond_C\,\star = \\
&(Tm : \mathsf{Ty}^+_{\mathcal{D}}\,(\mathsf{y}\,\diamond))) \times \mathsf{Tm}_{\mathcal{D}}\,\big(\mathsf{y}\,\diamond \triangleright (Tm \Rightarrow^+_{\mathcal{D}} Tm)\big)\,\big(Tm[\mathsf{p}]\big) \times \mathsf{Tm}_{\mathcal{D}}\,(\mathsf{y}\,\diamond \triangleright Tm)\,(Tm \Rightarrow Tm[\mathsf{p}]) = \\
&(Tm : (I : C) \to C(I, \diamond) \to \mathsf{Set}) \times (-[-]_{Tm} : Tm\,I\,\epsilon \to C(J,I) \to Tm\,J\,\epsilon) \times \ldots \times \\
&\quad (- \triangleright_{Tm} - : (I : C) \to C(I,\diamond) \to C) \times \cdots \times \big(lam : C(I,\diamond) \times Tm\,(I \triangleright_{Tm} \epsilon) \to Tm\,I\,\epsilon\big) \times \cdots \times \\
&\quad (app : C(I,\diamond) \times Tm\,I\,\epsilon \to (\{J : C\} \to C(J,I) \times Tm\,J\,\epsilon \to Tm\,J\,\epsilon) \times \ldots) \times \ldots
\end{aligned}
$$

As we can see, the naive semantics produces some encoding overhead: the above definition differs from Definition 4 in the following ways: the operations are uncurried, have several extra $C(I,\diamond)$ arguments (which can be all filled by ϵ), and the type of *app* quantifies over another object of C for each argument. This is the result of using the usual presheaf universe and function space for interpreting U and Π. We will rectify this in the next section.

5 Direct semantics of SOGAT signatures

In this section, we define first-order models of SOGATs using a more careful version of the presheaf model. We make sure that no Yoneda-encodings are present in the semantics using the idea of two-level type theory [4, 10] where presheaves over a CwF include a universe of "inner types" coming from the CwF. We extend two-level type theory with a separate function space where the domain is an inner type. This function space is isomorphic to the usual presheaf function space, but has a simpler semantics.

▶ **Problem 19** (Presheaves over a CwF). *If C is a CwF, then $\mathsf{PSh}(C)$ models ToS without using the usual presheaf U and Π.*

Construction. We interpret $\top$, Σ, Eq as in Problem 16, but define U, El and Π by Ty_C, Tm_C and $\triangleright_C$, respectively: $\mathsf{U}\,I\,\gamma_I := \mathsf{Ty}_C\,I$, $\mathsf{El}\,a\,I\,\gamma_I := \mathsf{Tm}_C\,I\,(a\,\gamma_I)$, $\Pi\,a\,B\,I\,\gamma_I := B\,(I \triangleright_C a\,\gamma_I)\,(\gamma_I[\mathsf{p}_C]_\Gamma, \mathsf{q}_C)$ with $\mathsf{lam}\,b\,\gamma_I := b\,(\gamma_I[\mathsf{p}_C]_\Gamma, \mathsf{q}_C)$ and $\mathsf{app}\,t\,(\gamma_I, a_I) := t\,\gamma_I[\mathsf{id}_I ,_C a_I]_B$. ◀

▶ **Problem 20** (Presheaves over a CwF^+). *If the category C is a CwF$^+$, then the previous model extends to a model of ToS$^+$ (Definition 15).*

Construction. We interpret U^+, el^+ and π^+ by Ty^+_C, identity and Π^+_C, respectively: $\mathsf{U}^+\,I\,\gamma_I := \mathsf{Ty}^+_C\,I$, $\mathsf{el}^+\,a\,\gamma_I := a\,\gamma_I$, $\pi^+\,a\,b\,\gamma_I := \Pi^+\,(a\,\gamma_I)\,(b\,(\gamma_I[\mathsf{p}_C]_\Gamma, \mathsf{q}_C))$, $\mathsf{lam}^+\,t\,\gamma_I := \mathsf{lam}^+_C\,(t\,(\gamma_I[\mathsf{p}_C]_\Gamma, \mathsf{q}_C))$, $\mathsf{app}^+\,t\,(\gamma_I, a_I) := \mathsf{app}^+_C\,(t\,\gamma_I)[\mathsf{id}_I\,,_C\,a_I]_{\mathsf{Tm}_C}$. ◀

▶ **Definition 21** (Direct semantics). *A direct model of a SOGAT signature Ω is a category C with a terminal object together with the interpretation of Ω in presheaves over presheaves over C, evaluated at the terminal presheaf: $(C : \mathsf{Cat}^\diamond) \times [\![\Omega]\!]_{\mathsf{PSh}(\mathsf{PSh}(C))}\ \diamond_{\mathsf{PSh}(C)}\ \star$. Note that this makes sense because $\mathsf{PSh}(C) : \mathsf{CwF}^+$, hence $\mathsf{PSh}(\mathsf{PSh}(C))$ is a model of ToS$^+$.*

We revisit the example from the end of the previous section. We again assume a $C : \mathsf{Cat}^\diamond$ and write $\mathcal{D} := \mathsf{PSh}(C)$ and $\mathcal{E} := \mathsf{PSh}(\mathcal{D})$.

$$
\begin{aligned}
&\left[\!\!\left[\Sigma\,\mathsf{U}^+\left((\mathsf{q} \Rightarrow^+ \mathsf{el}^+\,\mathsf{q}) \Rightarrow \mathsf{El}\,(\mathsf{el}^+\,\mathsf{q})\right) \times \left(\mathsf{el}^+\,\mathsf{q} \Rightarrow \mathsf{el}^+\,\mathsf{q} \Rightarrow \mathsf{El}\,(\mathsf{el}^+\,\mathsf{q})\right)\right)\right]\!\!\right]_{\mathcal{E}}\ \diamond_{\mathcal{D}}\ \star = \\
&(Tm : \mathsf{Ty}^+_{\mathcal{D}}\,\diamond_{\mathcal{D}}) \times \mathsf{Tm}_{\mathcal{D}}\left(\diamond \triangleright (Tm \Rightarrow^+_{\mathcal{D}} Tm)\right)\,(Tm[\mathsf{p}]) \times \mathsf{Tm}_{\mathcal{D}}\,(\diamond \triangleright Tm \triangleright Tm[\mathsf{p}])\,(Tm[\mathsf{p}][\mathsf{p}]) = \\
&(Tm : C \to \mathbb{1} \to \mathsf{Set}) \times (-[-]_{Tm} : Tm\,I\,\star \to C(J, I) \to Tm\,J\,\star) \times \ldots \times \\
&\quad (- \triangleright_{Tm} - : C \to \mathbb{1} \to C) \times \cdots \times \left(lam : \mathbb{1} \times Tm\,(I \triangleright_{Tm} \star) \to Tm\,I\,\star\right) \times \cdots \times \\
&\quad (app : \mathbb{1} \times Tm\,I\,\star \times Tm\,I\,\star \to Tm\,I\,\star) \times \ldots
\end{aligned}
$$

This translation is closer to computing Definition 4 from Definition 3: the only remaining noise is that the types of *Tm*, *lam* and *app* include extra $\mathbb{1}$ components and *app* is uncurried. In the next section, we will remove the extra $\mathbb{1}$s and make the type of application curried.

▶ **Theorem 22.** *For any signature, the naive and direct semantics result in isomorphic notions of models.*

Proof. We fix a $C : \mathsf{Cat}^\diamond$, and denote $\mathcal{D} := \mathsf{PSh}(C)$ and $\mathcal{E} := \mathsf{PSh}(\mathcal{D})$. $\mathcal{D}$ is a model of ToS$^+$ via Definition 18 and $\mathcal{E}$ is a model via Definition 21, and Yoneda navigates between them (it is not only a functor, but a CwF pseudomorphism [37]). By induction on the syntax of ToS$^+$, we define α for contexts, substitutions, types and terms: $\alpha_\Gamma : \mathsf{Sub}_{\mathcal{E}}\,[\![\Gamma]\!]_{\mathcal{E}}\,(\mathsf{y}\,[\![\Gamma]\!]_{\mathcal{D}})$, $\alpha_\gamma : \alpha_\Gamma \circ [\![\gamma]\!]_{\mathcal{E}} = \mathsf{y}\,[\![\gamma]\!]_{\mathcal{D}} \circ \alpha_\Delta$, $\alpha_A : [\![A]\!]_{\mathcal{E}} \cong \mathsf{y}\,[\![A]\!]_{\mathcal{D}}[\alpha_\Gamma]$, $\alpha_a : \alpha_A[\mathsf{id}, [\![a]\!]_{\mathcal{E}}] = \mathsf{y}\,[\![a]\!]_{\mathcal{D}}[\alpha_\Gamma]$. For a signature $\Omega : \mathsf{Ty}\,\diamond$, we thus obtain $[\![\Omega]\!]_{\mathcal{E}}\ \diamond_{\mathcal{D}}\ \star \cong \mathsf{y}\,[\![\Omega]\!]_{\mathcal{D}}[\alpha_\Gamma]\ \diamond_{\mathcal{D}}\ \star = \mathsf{Tm}_{\mathcal{D}}\ \diamond_{\mathcal{D}}\ [\![\Omega]\!]_{\mathcal{D}}$. ◀

Note that there is no size issue when stating the isomorphism because even if $\mathcal{E}$ is one level up compared to $\mathcal{D}$, we only use small components from $\mathcal{E}$ when evaluating into it.

6 GAT signature semantics of SOGAT signatures

In this section we translate SOGAT signatures into GAT signatures. The idea is the same as in the previous two sections: the GAT signature will start with a category with terminal object and then contain the presheaf interpretation of the SOGAT signature over that category. However now the presheaf model is not expressed in the metatheory, but internally to the theory of GAT signatures. This is challenging because this language is quite limited: there are no higher-order functions, no real universe, and so on.

In this section we work internally to presheaves over the syntax of ToS. Another way to say this is that we work in two-level type theory where the inner model is the syntax of ToS. Hence, we have the components $\mathsf{Ty} : \mathsf{Set}$, $\mathsf{Tm} : \mathsf{Ty} \to \mathsf{Set}$, $\ldots$, $\mathsf{refl} : (u = v) \cong \mathsf{Tm}\,(\mathsf{Eq}\,A\,u\,v) : \mathsf{reflect}$ of Definition 12 available (these are the inner types and type formers). We will build a first-order model of ToS$^+$, and the final result of the translation will be an element of Ty.

▶ **Construction 23** (Curried Π). *By induction-recursion, we define the* Σ*-closure of* U.

$$\begin{aligned}
&\mathsf{U}^* : \mathsf{Set} && \mathsf{El}^* : \mathsf{U}^* \to \mathsf{Ty}\\
&\top^* : \mathsf{U}^* && \mathsf{El}^*\,\top^* \quad := \top\\
&\Sigma^* : (as : \mathsf{U}^*) \to \left(\mathsf{Tm}\,(\mathsf{El}^*\,as) \to \mathsf{Tm}\,\mathsf{U}\right) \to \mathsf{U}^* && \mathsf{El}^*\,(\Sigma^*\,as\,b) := \Sigma\,(\mathsf{El}^*\,as)\,\lambda x.\mathsf{El}\,(b\,x)
\end{aligned}$$

By induction on U^**, we define the curried function space with* U^* *domain.*

$$\begin{aligned}
&\Pi^* : (as : \mathsf{U}^*) \to (\mathsf{Tm}\,(\mathsf{El}^*\,as) \to \mathsf{Ty}) \to \mathsf{Ty}\\
&\Pi^*\,\top^* \quad B := B\,\mathsf{tt}\\
&\Pi^*\,(\Sigma^*\,as\,c)\,B := \Pi^*\,as\,\left(\lambda xs.\Pi\,(c\,xs)\,\lambda y.B\,(xs,y)\right)
\end{aligned}$$

Π^* *comes with* lam^*, $\cdot^*$, *and* β, η *laws all defined by induction on* U^* *providing the following isomorphism.*

$$\mathsf{lam}^* : \left((xs : \mathsf{Tm}\,(\mathsf{El}^*\,as)) \to \mathsf{Tm}\,(B\,xs)\right) \cong \mathsf{Tm}\,(\Pi^*\,as\,B) : - \cdot^* -$$

We define the signature for categories with a terminal object by $\mathsf{Cat}^\diamond : \mathsf{Ty} := \Sigma\,\mathsf{U}\,\lambda Ob.\Sigma\,(Ob \Rightarrow Ob \Rightarrow \mathsf{U})\,\lambda Hom\ldots$ We assume a $C : \mathsf{Tm}\,\mathsf{Cat}^\diamond$, we refer to its components by Ob, Hom, ...

▶ **Problem 24** (A CwF^+ $\mathcal{D}$ of presheaves over C). *There is a notion of CwF*$^+$ *where the sorts of types and terms are* Ty*-valued. We construct such a CwF*$^+$ $\mathcal{D}$ *of presheaves over* C.

Construction. The category part is given by U^*-valued presheaves and natural transformations where $\mathsf{Con}_\mathcal{D} := \left(\Gamma : \mathsf{Tm}\,(\mathsf{El}\,Ob) \to \mathsf{U}^*\right) \times \left(-[-]_\Gamma : \mathsf{Tm}\left(\mathsf{El}^*\,(\Gamma\,I)\right) \to \mathsf{Tm}\left(\mathsf{El}\,(Hom{\cdot}J{\cdot}I)\right) \to \mathsf{Tm}\left(\mathsf{El}^*\,(\Gamma\,J)\right)\right)$×(functoriality) and $\mathsf{Sub}_\mathcal{D}\,\Delta\,\Gamma := \left(\gamma : \mathsf{Tm}\left(\mathsf{El}^*\,(\Delta\,I)\right) \to \mathsf{Tm}\left(\mathsf{El}^*\,(\Gamma\,I)\right)\right) \times$ (naturality). Recall that Ty and Tm are those of the syntax of ToS. We make sure that Ty, Tm have enough structure to define U-valued presheaves. For example, we define $\mathsf{Ty}_\mathcal{D} : \mathsf{Con}_\mathcal{D} \to \mathsf{Ty}$ by

$$\begin{aligned}
&\mathsf{Ty}_\mathcal{D}\,\Gamma := \Sigma\,(\Pi\,Ob\,\lambda I.\Gamma\,I \Rightarrow^* \mathsf{U})\,\lambda A.\Sigma\\
&\left(\Pi\,Ob\,\lambda I.\Pi^*\,(\Gamma\,I)\,\lambda\gamma_I.A\cdot I \cdot^* \gamma_I \Rightarrow \Pi\,Ob\,\lambda J.\Pi\,(Hom\cdot J\cdot I)\,\lambda f.\mathsf{El}\left(A\cdot J \cdot^* (\gamma_I[f]_\Gamma)\right)\right)\ldots
\end{aligned}$$

We define $\mathsf{Tm}_\mathcal{D} : (\Gamma : \mathsf{Con}_\mathcal{D}) \to \mathsf{Tm}\,(\mathsf{Ty}_\mathcal{D}\,\Gamma) \to \mathsf{Ty}$ as $\mathsf{Tm}_\mathcal{D}\,\Gamma\,A := \Sigma\left(\Pi\,Ob\,\lambda I.\Pi^*\,(\Gamma\,I)\,\lambda\gamma.\right.$ $\left.\mathsf{El}\,(A\cdot I \cdot^* \gamma)\right)\ldots$ Context extension $\triangleright_\mathcal{D}$ is Σ^*, $\mathsf{Ty}^+_\mathcal{D}$ is the same as $\mathsf{Ty}_\mathcal{D}$ extended with an $\triangleright_A$ operator in $\Pi\,Ob\,\lambda I.\Gamma\,I \Rightarrow^* \mathsf{El}\,Ob$, and its universal property. We define the first component of $\Pi^+_\mathcal{D} : \left(A : \mathsf{Tm}\,(\mathsf{Ty}^+_\mathcal{D}\,\Gamma)\right) \to \mathsf{Tm}\left(\mathsf{Ty}_\mathcal{D}\,(\Gamma \triangleright_\mathcal{D} A)\right) \to \mathsf{Tm}\,(\mathsf{Ty}_\mathcal{D}\,\Gamma)$ by $\Pi^+_\mathcal{D}\,A\,B\cdot I \cdot^* \gamma_I :=$ $B\cdot(\triangleright_A{\cdot}I \cdot^* \gamma_I) \cdot^* (\gamma_I[\mathsf{p}_A]_\Gamma, \mathsf{q}_A)$ where $\triangleright_A$, p_A and q_A are components in the input A. Note the careful distinguishing of metatheoretic function application, ·s and $\cdot^*$s. The full details are given as Supplementary Material. ◀

▶ **Problem 25** ($\mathcal{E} := \mathsf{PSh}(\mathcal{D})$). *The* Ty*-valued presheaves over* $\mathcal{D}$ *are a first-order model of ToS*$^+$. *We name this model* $\mathcal{E}$.

Proof. $\mathsf{Con}_\mathcal{E}$ is defined as $(\Psi : \mathsf{Con}_\mathcal{D} \to \mathsf{Ty}) \times (-[-]_\Psi : \mathsf{Tm}\,(\Psi\,\Gamma) \to \mathsf{Sub}_\mathcal{D}\,\Delta\,\Gamma \to \mathsf{Tm}\,(\Psi\,\Delta)) \times$ (functoriality). Types are Ty-valued dependent presheaves, terms are sections, context extension $\triangleright_\mathcal{E}$ and $\Sigma_\mathcal{E}$ are given by Σ. $\mathsf{U}_\mathcal{E}$, $\mathsf{El}_\mathcal{E}$, $\Pi_\mathcal{E}$ are given by $\mathsf{Ty}_\mathcal{D}$, $\mathsf{Tm}_\mathcal{D}$, $\triangleright_\mathcal{D}$, respectively. $\mathsf{Eq}_\mathcal{E}$ is pointwise Eq, its restriction operation and $\mathsf{reflect}_\mathcal{E}$ use reflect. $\mathsf{U}^+_\mathcal{E}$, $\mathsf{el}^+_\mathcal{E}$, $\pi^+_\mathcal{E}$ are defined by $\mathsf{Ty}^+_\mathcal{D}$, identity and $\Pi^+_\mathcal{D}$, respectively. ◀

▶ **Construction 26** (SOGAT → GAT translation)**.** *Given an* $\Omega : \mathsf{Ty}\,\diamond$ *in the first-order syntax of ToS*$^+$*, its GAT translation is* $\Sigma\,\mathsf{Cat}^\diamond\,\lambda\mathcal{C}.[\![\Omega]\!]_{\mathcal{E}(\mathcal{C})}\,\diamond_{\mathcal{D}(\mathcal{C})}\,\mathsf{tt}$ *where we explicitly marked that* $\mathcal{D}$ *and* $\mathcal{E}$ *depend on* $\mathcal{C}$*.*

Now we can reuse the semantics of GATs [43, Chapter 4] for any SOGAT, e.g. there is a category of models with an initial object, notions of dependent/displayed models, sections, induction is equivalent to initiality, free models, cofree models [45].

Our running example assuming $\mathcal{C} : \mathsf{Tm}\,\mathsf{Cat}^\diamond$ (its first two components named *Ob*, *Hom*):

$$
\begin{aligned}
&\left[\!\left[\Sigma\,\mathsf{U}^+\left(((\mathsf{q} \Rightarrow^+ \mathsf{el}^+\,\mathsf{q}) \Rightarrow \mathsf{El}\,(\mathsf{el}^+\,\mathsf{q})) \times (\mathsf{el}^+\,\mathsf{q} \Rightarrow \mathsf{el}^+\,\mathsf{q} \Rightarrow \mathsf{El}\,(\mathsf{el}^+\,\mathsf{q}))\right)\right]\!\right]_{\mathcal{E}}\,\diamond_{\mathcal{D}}\,\mathsf{tt} =\\
&\Sigma\,(\mathsf{Ty}^+_{\mathcal{D}}\,\diamond_{\mathcal{D}})\,\lambda\mathit{Tm}.\mathsf{Tm}_{\mathcal{D}}\left(\diamond \triangleright (\mathit{Tm} \Rightarrow^+_{\mathcal{D}} \mathit{Tm})\right)(\mathit{Tm}[\mathsf{p}]) \times \mathsf{Tm}_{\mathcal{D}}\,(\diamond \triangleright \mathit{Tm} \triangleright \mathit{Tm}[\mathsf{p}])\,(\mathit{Tm}[\mathsf{p}][\mathsf{p}]) =\\
&\Sigma\Big(\Sigma\,(\mathit{Ob} \Rightarrow \mathsf{U})\,\lambda\mathit{Tm}.\Sigma\,(\Pi\,\mathit{Ob}\,\lambda I.\mathit{Tm}\cdot I \Rightarrow \Pi\,\mathit{Ob}\,\lambda J.\mathit{Hom}\cdot J\cdot I \Rightarrow \mathsf{El}\,(\mathit{Tm}\cdot J))\ldots\\
&\quad\Sigma\,(\mathit{Ob} \Rightarrow \mathsf{El}\,\mathit{Ob})\ \ldots\Big)\,\lambda(\mathit{Tm},\ldots,\triangleright_{\mathit{Tm}},\ldots).\Sigma\left(\Sigma\,(\Pi\,\mathit{Ob}\,\lambda I.\mathit{Tm}\cdot(\triangleright\cdot I) \Rightarrow \mathsf{El}\,(\mathit{Tm}\cdot I))\ \ldots\right)\\
&\qquad\lambda\mathit{lam}.\Sigma(\Pi\,\mathit{Ob}\,\lambda I.\mathit{Tm}\cdot I \Rightarrow \mathit{Tm}\cdot I \Rightarrow \mathsf{El}\,(\mathit{Tm}\cdot I))\ldots
\end{aligned}
$$

The second line is the same as for the direct semantics, but now $\mathcal{D}$ is defined using the curried function space, which removes the extra $\mathbb{1}$s and makes application curried when we unfold even more. As we now compute a formal signature in Ty, we do not use implicit arguments, and use λ for binders. The only difference from Definition 4 is that the components for $\mathsf{Cat}^\diamond$, *Tm* and *lam* are separate (flat) Σ types, rather than one flat iterated Σ.

We implemented the SOGAT → GAT translation in Agda using partial deep embeddings of ToS and ToS$^+$. It computes the expected GAT signatures for a number of SOGAT examples. It is available as Supplementary Material.

The GAT semantics was defined relative to the syntax of ToS. However, it works for any model of ToS: if we use the standard model of ToS (set model, metacircular interpretation where $\mathsf{Con} = \mathsf{Set}$, $\mathsf{Ty}\,\Gamma = \Gamma \to \mathsf{Set}$, $\mathsf{Tm}\,\Gamma\,A = (\gamma : \Gamma) \to A\,\gamma$) instead of the syntax, we obtain another notion of model for each SOGAT signature. We show that this notion of model is isomorphic to the direct semantics from the previous section.

▶ **Theorem 27.** *For any SOGAT signature, the direct semantics and the GAT semantics over the standard model yield isomorphic notions of models.*

Proof. We work in presheaves over the standard model of ToS. We observe that in this model U and Ty are Russell-universes and are closed under type formers Σ, Π, Eq without the restrictions we have in the syntax of ToS. We reformulate Definition 21 in this internal language: the category $\mathcal{C}$ becomes an element of $\mathsf{Tm}\,\mathsf{Cat}^\diamond$, the $\mathcal{D}' := \mathsf{PSh}(\mathcal{C})$ is a CwF$^+$ with Ty-valued types and terms. We compare this $\mathcal{D}'$ and the $\mathcal{D}$ given by Problem 24: we define $\alpha : \mathcal{D} \to \mathcal{D}'$ as a strict CwF$^+$-morphism which is bijective on Ty, Ty^+ and Tm. The content of α is mapping in and out of the inductive-recursive universe U^*. We denote $\mathcal{E} := \mathsf{PSh}(\mathcal{D})$ and $\mathcal{E}' := \mathsf{PSh}(\mathcal{D}')$. Precomposition with α is $\alpha^* : \mathsf{PSh}(\mathcal{D}') \to \mathsf{PSh}(\mathcal{D})$ which is a strict CwF-morphism. Now, by induction on the syntax of ToS$^+$, we define β for contexts, substitutions, types and terms: $\beta_\Gamma : \mathsf{Sub}_{\mathcal{E}}\,[\![\Gamma]\!]_{\mathcal{E}}\,(\alpha^*\,[\![\Gamma]\!]_{\mathcal{E}'})$, $\beta_\gamma : \beta_\Gamma \circ [\![\gamma]\!]_{\mathcal{E}} = \alpha^*\,[\![\gamma]\!]_{\mathcal{E}'} \circ \beta_\Delta$, $\beta_A : [\![A]\!]_{\mathcal{E}} \cong \alpha^*\,[\![A]\!]_{\mathcal{E}'}[\beta_\Gamma]$, $\beta_a : \beta_A[\mathsf{id}, [\![a]\!]_{\mathcal{E}}] = \alpha^*\,[\![a]\!]_{\mathcal{E}'}[\beta_\Gamma]$. Now for a signature $\Omega : \mathsf{Ty}\,\diamond$, from β_Ω we have $[\![\Omega]\!]_{\mathcal{E}}\,\diamond_{\mathcal{D}}\,\star \cong \alpha^*\,[\![\Omega]\!]_{\mathcal{E}'}[\beta_\diamond]\,\diamond_{\mathcal{D}}\,\star = [\![\Omega]\!]_{\mathcal{E}'}\,\diamond_{\mathcal{D}'}\,\star$. ◀

▶ **Corollary 28.** *By combining the isomorphisms of Theorems 22 and 27: for any SOGAT signature, in presheaves over any of its first-order models, a second-order model is available.*

This corollary formalises the diagonal internalisation arrow $\mathcal{S} \longmapsto \mathbb{S}$ in [16, page 3].

7 Extensions and variants

In this section, we sketch some extensions of ToS$^+$ and the alternative single substitution calculus semantics, see the Supplementary Material for details.

Open and infinitary SOGATs. The SOGAT→GAT translation also works in the case when signatures are open (can refer to external types like $\mathbb{N}$ in Definition 10) or infinitary. In this case the theory of signatures is defined in the outer layer of a two-level type theory where the inner layer is any chosen CwF, and signatures can refer to the universe Set° of inner types [43, Chapter 3]. The theory of possibly open signatures includes a type former $\hat{\Pi} : (A : \mathsf{Set}^\circ) \to (A \to \mathsf{Ty}) \to \mathsf{Ty}$, with the universal property $\big((a : A) \to \mathsf{Tm}\,(B\,a)\big) \cong \mathsf{Tm}\,(\hat{\Pi}\,A\,B)$. For example, Definition 10 is formalised as $\Sigma\,(\mathbb{N} \mathbin{\hat{\Rightarrow}} \mathsf{U})\lambda\,Ty.\Sigma\,(\hat{\Pi}\,\mathbb{N}\,\lambda i.\mathsf{Ty}\mathbin{\hat{}} i \Rightarrow \mathsf{U})\lambda\,Tm\ldots$ where $\mathbb{N} : \mathsf{Set}^\circ$. Similarly, for infinitary signatures, we have a type former $\tilde{\pi} : (A : \mathsf{Set}^\circ) \to (A \to \mathsf{Tm}\,\mathsf{U}) \to \mathsf{Tm}\,\mathsf{U}$ with the universal property $\big((a : A) \to \mathsf{Tm}\,(\mathsf{El}\,(b\,a))\big) \cong \mathsf{Tm}\,(\mathsf{El}\,(\tilde{\pi}\,A\,b))$. When supporting infinitary operations, we have to replace the general Eq type by an equality of types in U. This is because the semantics of infinitary GATs is not compatible with sort equations [43, Chapter 5].

Semantics using single substitution calculus. Our translation from SOGAT to GAT is not canonical: for example, we could have used semicategories instead of categories. There is also a minimalistic version of the translation which results in a single substitution calculus (SSC), which does not involve a category (single substitutions are not composable). For the SOGAT given by the signature $\Sigma\,\mathsf{U}\,\lambda Ty.Ty \Rightarrow \mathsf{U}^+$, the parallel translation results in the GAT known as CwF. The SSC translation for the same SOGAT gives a smaller theory: there is no composition of substitutions, no identity substitution, no empty substitution ϵ and no $-,-$ operator for building substitutions into extended contexts. We have single weakening $\mathsf{p} : \mathsf{Sub}\,(\Gamma \triangleright A)\,\Gamma$, single substitution $\langle - \rangle : \mathsf{Tm}\,\Gamma\,A \to \mathsf{Sub}\,\Gamma\,(\Gamma \triangleright A)$ and a lifting operation on substitutions $-^+ : (\gamma : \mathsf{Sub}\,\Delta\,\Gamma) \to \mathsf{Sub}\,(\Delta \triangleright A[\gamma])\,(\Gamma \triangleright A)$. There are four equations for types: $A[\mathsf{p}][\gamma^+] = A[\gamma][\mathsf{p}]$, $A[\mathsf{p}][\langle b \rangle] = A$, $A[\langle b \rangle][\gamma] = A[\gamma^+][\langle b[\gamma] \rangle]$, $A[\mathsf{p}^+][\langle \mathsf{q} \rangle] = A$ and four equations for terms: $\mathsf{q}[\langle b \rangle] = b$, $\mathsf{q}[\gamma^+] = \mathsf{q}$, $b[\mathsf{p}][\gamma^+] = b[\gamma][\mathsf{p}]$, $b[\mathsf{p}][\langle a \rangle] = b$. The resulting theory is a minimalistic variant of B systems [1]. CwFs are models of the resulting theory, but not the other way.[1] The syntaxes are however equivalent [41]. This situation is analogous to the relationship of lambda calculus and combinatory logic [8], where combinatory logic has more models, but the sets of syntactic terms are isomorphic.

With small modifications, the translation described in Section 6 can be used to obtain the SSC translation of a GAT. We only change the construction for Problem 24: $\mathcal{C}$ is not a category, just a graph with a vertex $\diamond$; $\mathsf{Con}_\mathcal{D}$ and $\mathsf{Ty}_\mathcal{D}$ do not include functoriality equations; $A : \mathsf{Ty}^+_\mathcal{D}\,\Gamma$ includes $\triangleright_A$, but not the usual universal property; instead we have p_A, q_A, $\langle - \rangle_A$, $-^{+_A}$ operations and the above described 8 equations.

8 Conclusions and further work

In this paper we described SOGAT signatures and translations from SOGAT signatures to GAT signatures. Correctness of our parallel substitution-based translation was shown by constructing an isomorphism with the naive semantics, and was validated by several examples.

[1] We can restrict any CwF to be only an SSC: we build a new sort of substitutions out of a single term or a single weakening inductively. These substitutions do not compose, so they do not form a category, but they form a model of the above described SSC.

In the future we would like to show equivalence with Uemura's semantic definition of SOGATs. We would like to computer check our constructions possibly using strict presheaves [46]. It would be interesting to understand the exact relationship between our parallel and single substitution calculi: we conjecture that for any SOGAT, they yield equivalent syntaxes.

We hope that our paper makes a step towards proof assistants with SOGAT support. In such a system, the user could specify the signature for a SOGAT using a built-in ToS$^+$, and would automatically obtain notions of first-order and second-order models, morphisms, iterators, induction principles (also for second-order displayed models [16]), and so on.

References

1 Benedikt Ahrens, Jacopo Emmenegger, Paige Randall North, and Egbert Rijke. B-systems and C-systems are equivalent. *The Journal of Symbolic Logic*, pages 1–9, 2023. doi:10.1017/jsl.2023.41.

2 Benedikt Ahrens, André Hirschowitz, Ambroise Lafont, and Marco Maggesi. Modular specification of monads through higher-order presentations. In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany*, volume 131 of *LIPIcs*, pages 6:1–6:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.FSCD.2019.6.

3 Guillaume Allais, Robert Atkey, James Chapman, Conor McBride, and James McKinna. A type- and scope-safe universe of syntaxes with binding: their semantics and proofs. *J. Funct. Program.*, 31:e22, 2021. doi:10.1017/S0956796820000076.

4 Thorsten Altenkirch, Paolo Capriotti, and Nicolai Kraus. Extending homotopy type theory with strict equality. In Jean-Marc Talbot and Laurent Regnier, editors, *25th EACSL Annual Conference on Computer Science Logic, CSL 2016, August 29 - September 1, 2016, Marseille, France*, volume 62 of *LIPIcs*, pages 21:1–21:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICS.CSL.2016.21.

5 Thorsten Altenkirch, Yorgo Chamoun, Ambrus Kaposi, and Michael Shulman. Internal parametricity, without an interval. *Proc. ACM Program. Lang.*, 8(POPL):2340–2369, 2024. doi:10.1145/3632920.

6 Thorsten Altenkirch and Ambrus Kaposi. Normalisation by evaluation for dependent types. In Delia Kesner and Brigitte Pientka, editors, *1st International Conference on Formal Structures for Computation and Deduction, FSCD 2016, June 22-26, 2016, Porto, Portugal*, volume 52 of *LIPIcs*, pages 6:1–6:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICS.FSCD.2016.6.

7 Thorsten Altenkirch and Ambrus Kaposi. Type theory in type theory using quotient inductive types. In Rastislav Bodík and Rupak Majumdar, editors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 18–29. ACM, 2016. doi:10.1145/2837614.2837638.

8 Thorsten Altenkirch, Ambrus Kaposi, Artjoms Sinkarovs, and Tamás Végh. Combinatory logic and lambda calculus are equal, algebraically. In Marco Gaboardi and Femke van Raamsdonk, editors, *8th International Conference on Formal Structures for Computation and Deduction, FSCD 2023, July 3-6, 2023, Rome, Italy*, volume 260 of *LIPIcs*, pages 24:1–24:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.FSCD.2023.24.

9 Thorsten Altenkirch and Bernhard Reus. Monadic presentations of lambda terms using generalized inductive types. In Jörg Flum and Mario Rodríguez-Artalejo, editors, *Computer Science Logic, 13th International Workshop, CSL '99, 8th Annual Conference of the EACSL, Madrid, Spain, September 20-25, 1999, Proceedings*, volume 1683 of *Lecture Notes in Computer Science*, pages 453–468. Springer, 1999. doi:10.1007/3-540-48168-0_32.

10 Danil Annenkov, Paolo Capriotti, Nicolai Kraus, and Christian Sattler. Two-level type theory and applications. *Mathematical Structures in Computer Science*, 33(8):688–743, 2023. doi:10.1017/S0960129523000130.

11 Samy Avrillon. Logic as a second-order generalized algebraic theory, 2023. Report on the 3-month research internship at the Faculty of Informatics of ELTE. URL: https://github.com/MysaaJava/m1-internship/releases/download/project-report/Avrillon-02.pdf.

12 Henk Barendregt. Introduction to generalized type systems. *Journal of Functional Programming*, 1(2):125–154, 1991. doi:10.1017/S0956796800020025.

13 Andrej Bauer, Philipp G. Haselwarter, and Peter LeFanu Lumsdaine. A general definition of dependent type theories. *CoRR*, abs/2009.05539, 2020. arXiv:2009.05539.

14 Rafaël Bocquet. External univalence for second-order generalized algebraic theories. *CoRR*, abs/2211.07487, 2022. doi:10.48550/arXiv.2211.07487.

15 Rafaël Bocquet. Towards coherence theorems for equational extensions of type theories. *CoRR*, abs/2304.10343, 2023. doi:10.48550/arXiv.2304.10343.

16 Rafaël Bocquet, Ambrus Kaposi, and Christian Sattler. For the metatheory of type theory, internal sconing is enough. In Marco Gaboardi and Femke van Raamsdonk, editors, *8th International Conference on Formal Structures for Computation and Deduction, FSCD 2023, July 3-6, 2023, Rome, Italy*, volume 260 of *LIPIcs*, pages 18:1–18:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.FSCD.2023.18.

17 Paolo Capriotti. Notions of type formers. In Ambrus Kaposi, editor, *23rd International Conference on Types for Proofs and Programs, TYPES 2017*. Eötvös Loránd University, 2017. URL: http://types2017.elte.hu/proc.pdf#page=77.

18 John Cartmell. Generalised algebraic theories and contextual categories. *Ann. Pure Appl. Log.*, 32:209–243, 1986. doi:10.1016/0168-0072(86)90053-9.

19 Simon Castellan, Pierre Clairambault, and Peter Dybjer. Categories with families: Unityped, simply typed, and dependently typed. *CoRR*, abs/1904.00827, 2019. arXiv:1904.00827.

20 Thierry Coquand. Canonicity and normalization for dependent type theory. *Theor. Comput. Sci.*, 777:184–191, 2019. doi:10.1016/J.TCS.2019.01.015.

21 Marcelo P. Fiore and Chung-Kil Hur. Second-order equational logic (extended abstract). In Anuj Dawar and Helmut Veith, editors, *Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23-27, 2010. Proceedings*, volume 6247 of *Lecture Notes in Computer Science*, pages 320–335. Springer, 2010. doi:10.1007/978-3-642-15205-4_26.

22 Marcelo P. Fiore, Andrew M. Pitts, and S. C. Steenkamp. Quotients, inductive types, and quotient inductive types. *Log. Methods Comput. Sci.*, 18(2), 2022. doi:10.46298/LMCS-18(2:15)2022.

23 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. IEEE Computer Society, 1999. doi:10.1109/LICS.1999.782615.

24 Jonas Frey. Duality for clans: a refinement of gabriel-ulmer duality. *CoRR*, abs/2308.11967, 2023. doi:10.48550/arXiv.2308.11967.

25 Gaëtan Gilbert, Jesper Cockx, Matthieu Sozeau, and Nicolas Tabareau. Definitional proof-irrelevance without K. *Proc. ACM Program. Lang.*, 3(POPL):3:1–3:28, 2019. doi:10.1145/3290316.

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

27 Robert Harper. *Practical Foundations for Programming Languages (2nd. Ed.)*. Cambridge University Press, 2016. URL: https://www.cs.cmu.edu/%7Erwh/pfpl/index.html.

28 Robert Harper. An equational logical framework for type theories. *CoRR*, abs/2106.01484, 2021. arXiv:2106.01484.

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

30 Philipp G. Haselwarter and Andrej Bauer. Finitary type theories with and without contexts. *J. Autom. Reason.*, 67(4):36, 2023. doi:10.1007/S10817-023-09678-Y.

31 Martin Hofmann. Syntax and semantics of dependent types. In *Semantics and Logics of Computation*, pages 79–130. Cambridge University Press, 1997.

32 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.

33 Jasper Hugunin. Why not W? In Ugo de'Liguoro, Stefano Berardi, and Thorsten Altenkirch, editors, *26th International Conference on Types for Proofs and Programs, TYPES 2020, March 2-5, 2020, University of Turin, Italy*, volume 188 of *LIPIcs*, pages 8:1–8:9. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.TYPES.2020.8.

34 Jonas Kaiser, Steven Schäfer, and Kathrin Stark. Binder aware recursion over well-scoped de Bruijn syntax. In June Andronick and Amy P. Felty, editors, *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2018, Los Angeles, CA, USA, January 8-9, 2018*, pages 293–306. ACM, 2018. doi:10.1145/3167098.

35 Ambrus Kaposi. Formalisation of type checking into algebraic syntax. https://bitbucket.org/akaposi/tt-in-tt/src/master/Typecheck.agda, 2018.

36 Ambrus Kaposi. Quotient inductive-inductive types and higher friends. Talk given at the Homotopy Type Theory Electronic Seminar Talks (HoTTEST), October 2020. URL: https://akaposi.github.io/pres_hottest.pdf.

37 Ambrus Kaposi, Simon Huber, and Christian Sattler. Gluing for type theory. In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany*, volume 131 of *LIPIcs*, pages 25:1–25:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.FSCD.2019.25.

38 Ambrus Kaposi and András Kovács. A syntax for higher inductive-inductive types. In Hélène Kirchner, editor, *3rd International Conference on Formal Structures for Computation and Deduction, FSCD 2018, July 9-12, 2018, Oxford, UK*, volume 108 of *LIPIcs*, pages 20:1–20:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.FSCD.2018.20.

39 Ambrus Kaposi, András Kovács, and Thorsten Altenkirch. Constructing quotient inductive-inductive types. *Proc. ACM Program. Lang.*, 3(POPL):2:1–2:24, 2019. doi:10.1145/3290315.

40 Ambrus Kaposi, András Kovács, and Ambroise Lafont. For finitary induction-induction, induction is enough. In Marc Bezem and Assia Mahboubi, editors, *25th International Conference on Types for Proofs and Programs, TYPES 2019, June 11-14, 2019, Oslo, Norway*, volume 175 of *LIPIcs*, pages 6:1–6:30. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.TYPES.2019.6.

41 Ambrus Kaposi and Szumi Xie. Type theory in type theory using single substitutions. In *30th International Conference on Types for Proofs and Programs*. IT University of Copenhagen, 2024.

42 András Kovács and Ambrus Kaposi. Large and infinitary quotient inductive-inductive types. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 648–661. ACM, 2020. doi:10.1145/3373718.3394770.

43 András Kovács. *Type-Theoretic Signatures for Algebraic Theories and Inductive Types*. PhD thesis, Eötvös Loránd University, Hungary, 2022. arXiv:2302.08837.

44 Paul Blain Levy, John Power, and Hayo Thielecke. Modelling environments in call-by-value programming languages. *Inf. Comput.*, 185(2):182–210, 2003. doi:10.1016/S0890-5401(03)00088-9.

45 Hugo Moeneclaey. Parametricity and semi-cubical types. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–11. IEEE, 2021. doi:10.1109/LICS52264.2021.9470728.

46 Pierre-Marie Pédrot. Russian constructivism in a prefascist theory. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 782–794. ACM, 2020. doi:10.1145/3373718.3394740.

47 Brigitte Pientka and Jana Dunfield. Beluga: A framework for programming and reasoning with deductive systems (system description). In Jürgen Giesl and Reiner Hähnle, editors, *Automated Reasoning*, pages 15–21, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

48 Benjamin C. Pierce. *Types and programming languages*. MIT Press, 2002.

49 Benjamin C. Pierce, Arthur Azevedo de Amorim, Chris Casinghino, Marco Gaboardi, Michael Greenberg, Cătălin Hriţcu, Vilhelm Sjöberg, Andrew Tolmach, and Brent Yorgey. *Programming Language Foundations*, volume 2 of *Software Foundations*. Electronic textbook, 2024. Version 6.5, http://softwarefoundations.cis.upenn.edu.

50 Jonathan Sterling. *First Steps in Synthetic Tait Computability: The Objective Metatheory of Cubical Type Theory*. PhD thesis, Carnegie Mellon University, USA, 2022. doi:10.1184/R1/19632681.V1.

51 Jonathan Sterling and Carlo Angiuli. Normalization for cubical type theory. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–15. IEEE, 2021. doi:10.1109/LICS52264.2021.9470719.

52 Taichi Uemura. *Abstract and Concrete Type Theories*. PhD thesis, University of Amsterdam, 2021.

53 Philip Wadler, Wen Kokke, and Jeremy G. Siek. *Programming Language Foundations in Agda*. Online, August 2022. URL: https://plfa.inf.ed.ac.uk/22.08/.

A More examples of languages as SOGATs

▶ **Definition 29** (Hindley–Milner type system)**.**

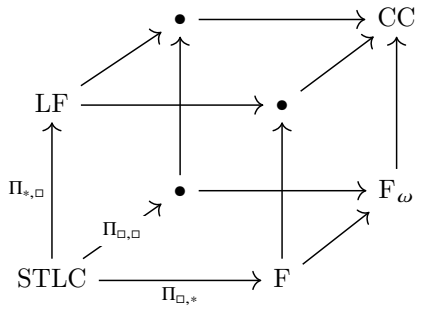
$$
\begin{array}{lll}
\mathsf{MTy} : \mathsf{Set} & & - \Rightarrow - : \mathsf{MTy} \to \mathsf{MTy} \to \mathsf{MTy} \\
\mathsf{Ty} \quad : \mathsf{Set} & \mathsf{lam} & : \big(\mathsf{Tm}\,(\mathsf{i}\,A) \to \mathsf{Tm}\,(\mathsf{i}\,B)\big) \cong \mathsf{Tm}\,(\mathsf{i}\,(A \Rightarrow B)) : - \cdot - \\
\mathsf{i} \quad : \mathsf{MTy} \to \mathsf{Ty} & \forall & : (\mathsf{MTy} \to \mathsf{Ty}) \to \mathsf{Ty} \\
\mathsf{Tm} \quad : \mathsf{Ty} \to \mathsf{Set} & \mathsf{Lam} & : \big((A : \mathsf{MTy}) \to \mathsf{Tm}\,(B\,A)\big) \cong \mathsf{Tm}\,(\forall\,B) : - \bullet -
\end{array}
$$

The language of fine-grain call by value is to Freyd categories [44] as simply typed lambda calculus is to cartesian closed categories. Here we add some type formers and a fixpoint operator for illustration. All variables are values (in Val).

▶ **Definition 30** (Fine-grain call by value)**.**

$$
\begin{array}{lll}
\mathsf{Ty} & : \mathsf{Set} & -\Rightarrow- : \mathsf{Ty} \to \mathsf{Ty} \to \mathsf{Ty} \\
\mathsf{Val} & : \mathsf{Ty} \to \mathsf{Set} & \mathsf{lam} \quad : (\mathsf{Val}\, A \to \mathsf{Tm}\, B) \to \mathsf{Val}\,(A \Rightarrow B) \\
\mathsf{Tm} & : \mathsf{Ty} \to \mathsf{Set} & -\cdot- \quad : \mathsf{Val}\,(A \Rightarrow B) \to \mathsf{Val}\, A \to \mathsf{Tm}\, B \\
\mathsf{return} & : \mathsf{Val}\, A \to \mathsf{Tm}\, A & \Rightarrow\beta \quad : \mathsf{lam}\, f \cdot a = f\, a \\
-\gg\!= - & : \mathsf{Tm}\, A \to (\mathsf{Val}\, A \to \mathsf{Tm}\, B) \to \mathsf{Tm}\, B & -\times- \quad : \mathsf{Ty} \to \mathsf{Ty} \to \mathsf{Ty} \\
\mathsf{idl} & : \mathsf{return}\, a \gg\!= f = f\, a & -,- \quad : \mathsf{Val}\, A \to \mathsf{Val}\, B \to \mathsf{Val}\,(A \times B) \\
\mathsf{idr} & : m \gg\!= \mathsf{return} = m & \mathsf{case}\times \quad : \mathsf{Val}\,(A \times B) \to \\
\mathsf{ass} & : (m \gg\!= f) \gg\!= g = & \qquad (\mathsf{Val}\, A \to \mathsf{Val}\, B \to \mathsf{Tm}\, C) \to \mathsf{Tm}\, C \\
 & \quad m \gg\!= (\lambda a.f\, a \gg\!= g) & \times\beta \quad : \mathsf{case}\times\,(a, b)\, f = f\, a\, b \\
\mathsf{T} & : \mathsf{Ty} \to \mathsf{Ty} & \mathsf{fix} \quad : (\mathsf{Val}\,(\mathsf{T}\, A) \to \mathsf{Tm}\, A) \to \mathsf{Tm}\, A \\
\mathsf{thunk} & : \mathsf{Tm}\, A \cong \mathsf{Val}\,(\mathsf{T}\, A) : \mathsf{force} & \mathsf{fix}\beta \quad : \mathsf{fix}\, f = f\,(\mathsf{thunk}\,(\mathsf{fix}\, f))
\end{array}
$$

The following definition shows that all the languages in the lambda cube [12] can be given as SOGATs. The simply typed lambda calculus (STLC) only includes $\Pi_{*,*}$, and the edges in each dimension add one of the other three Π types, respectively. The calculus of constructions (CC) includes all four Π types.



We don't give names to the maps in the universal properties.

▶ **Definition 31** (CC)**.**

$$
\begin{array}{lll}
\square & : \mathsf{Set} & \\
\mathsf{Ty} & : \square \to \mathsf{Set} & \\
* & : \square & \\
\mathsf{Tm} & : \mathsf{Ty}\, * \to \mathsf{Set} & \\
\Pi_{*,*} & : (A : \mathsf{Ty}\, *) \to (\mathsf{Tm}\, A \to \mathsf{Ty}\,(*)) \to \mathsf{Ty}\, * & \mathsf{Tm}\,(\Pi_{*,*}\, A\, B) \cong (a : \mathsf{Tm}\, A) \to \mathsf{Tm}\,(B\, a) \\
\Pi_{*,\square} & : (A : \mathsf{Ty}\, *) \to (\mathsf{Tm}\, A \to \square) \to \square & \mathsf{Ty}\,(\Pi_{*,\square}\, A\, L) \cong (a : \mathsf{Tm}\, A) \to \mathsf{Ty}\,(L\, a) \\
\Pi_{\square,*} & : (K : \square) \to (\mathsf{Ty}\, K \to \mathsf{Ty}\, *) \to \mathsf{Ty}\, * & \mathsf{Tm}\,(\Pi_{\square,*}\, K\, B) \cong (A : \mathsf{Ty}\, K) \to \mathsf{Tm}\,(B\, A) \\
\Pi_{\square,\square} & : (K : \square) \to (\mathsf{Ty}\, K \to \square) \to \square & \mathsf{Ty}\,(\Pi_{\square,\square}, K\, L) \cong (A : \mathsf{Ty}\, K) \to \mathsf{Ty}\,(L\, A)
\end{array}
$$

The next definition adds Σ, 0, 1, 2 and W-types to minimal Martin-Löf type theory, which is enough to encode all inductive types [33].

▶ **Definition 32** (Martin-Löf type theory with inductive types)**.** *We extend Definition 10 with the following.*

$$
\begin{array}{ll}
\Sigma & : (A : \mathsf{Ty}\, i) \to (\mathsf{Tm}\, A \to \mathsf{Ty}\, i) \to \mathsf{Ty}\, i \\
(-,-) & : (a : \mathsf{Tm}\, A) \times \mathsf{Tm}\, (B\, a) \cong \mathsf{Tm}\, (\Sigma\, A\, B) : \mathsf{fst}, \mathsf{snd} \\
\bot & : \mathsf{Ty}\, 0 \\
\mathsf{exfalso} & : \mathsf{Tm}\, \bot \to \mathsf{Tm}\, A \\
\top & : \mathsf{Ty}\, 0 \\
\mathsf{tt} & : \top \cong \mathsf{Tm}\, \top \\
\mathsf{Bool} & : \mathsf{Ty}\, 0 \\
\mathsf{true} & : \mathsf{Tm}\, \mathsf{Bool} \\
\mathsf{false} & : \mathsf{Tm}\, \mathsf{Bool} \\
\mathsf{indBool} & : (C : \mathsf{Tm}\, \mathsf{Bool} \to \mathsf{Ty}\, i) \to \mathsf{Tm}\, (C\, \mathsf{true}) \to \mathsf{Tm}\, (C\, \mathsf{false}) \to \\
& \quad (b : \mathsf{Tm}\, \mathsf{Bool}) \to \mathsf{Tm}\, (C\, b) \\
\mathsf{Bool}\beta_1 & : \mathsf{indBool}\, t\, f\, \mathsf{true} = t \\
\mathsf{Bool}\beta_2 & : \mathsf{indBool}\, t\, f\, \mathsf{false} = f \\
\mathsf{Id} & : (A : \mathsf{Ty}\, i) \to \mathsf{Tm}\, A \to \mathsf{Tm}\, A \to \mathsf{Ty}\, i \\
\mathsf{refl} & : (a : \mathsf{Tm}\, A) \to \mathsf{Tm}\, (\mathsf{Id}\, a\, a) \\
\mathsf{J} & : \big(C : (x : \mathsf{Tm}\, A) \to \mathsf{Tm}\, (\mathsf{Id}\, A\, a\, x) \to \mathsf{Ty}\, i\big) \to \\
& \quad \mathsf{Tm}\, \big(C\, a\, (\mathsf{refl}\, a)\big) \to (x : \mathsf{Tm}\, A)(e : \mathsf{Tm}\, (\mathsf{Id}\, A\, a\, x)) \to \mathsf{Tm}\, (C\, x\, e) \\
\mathsf{Id}\beta & : \mathsf{J}\, C\, w\, a\, (\mathsf{refl}\, a) = w \\
\mathsf{W} & : (S : \mathsf{Ty}\, i) \to (\mathsf{Tm}\, S \to \mathsf{Ty}\, i) \to \mathsf{Ty}\, i \\
\mathsf{sup} & : (s : \mathsf{Tm}\, S) \to (\mathsf{Tm}\, (P\, s) \to \mathsf{Tm}\, (\mathsf{W}\, S\, P)) \to \mathsf{Tm}\, (\mathsf{W}\, S\, P) \\
\mathsf{indW} & : (C : \mathsf{Tm}\, (\mathsf{W}\, S\, P) \to \mathsf{Ty}\, i) \to \\
& \quad \Big(\big((p : \mathsf{Tm}\, (P\, s)) \to \mathsf{Tm}\, (C\, (f\, p))\big) \to \mathsf{Tm}\, \big(C\, (\mathsf{sup}\, s\, f)\big)\Big) \to \\
& \quad (w : \mathsf{Tm}\, (\mathsf{W}\, \mathsf{S}\, \mathsf{P})) \to \mathsf{Tm}\, \big(C\, w\big) \\
\mathsf{W}\beta & : \mathsf{indW}\, C\, h\, (\mathsf{sup}\, s\, f) = h\, (\lambda p.\mathsf{indW}\, C\, h\, (f\, p))
\end{array}
$$

In the following example, we add a new sort of telescopes to type theory. This can also be seen as an inductive-recursive definition internally to presheaves over the syntax (or any model; it shows that any CwF with $\top$ and Σ can be extended with telescopes).

▶ **Definition 33** (Telescopes in Martin-Löf type theory). *We extend Definition 32 with the following.*

$$
\begin{array}{lll}
\mathsf{Tys} : \mathsf{Set} & \diamond : \mathsf{Tys} & -\triangleright- : (\overline{A} : \mathsf{Tys}) \to (\mathsf{Tms}\, \overline{A} \to \mathsf{Ty}) \to \mathsf{Tys} \\
\ulcorner - \urcorner : \mathsf{Tys} \to \mathsf{Ty} & \diamond\beta : \ulcorner \diamond \urcorner = \top & \triangleright\beta : \ulcorner \overline{A} \triangleright A \urcorner = \Sigma\, \ulcorner \overline{A} \urcorner\, A
\end{array}
$$

The next SOGAT has both telescopes and telecopic terms, it does not rely on the presence of Σ types.

▶ **Definition 34** (Telescopes and telescopic terms in Martin-Löf type theory). *We extend Definition 10 or Definition 32 with the following.*

$$
\begin{array}{lll}
\mathsf{Tys} : \mathsf{Set} & \diamond : \mathsf{Tys} & -\triangleright- : (\overline{A} : \mathsf{Tys}) \to (\mathsf{Tms}\, \overline{A} \to \mathsf{Ty}) \to \mathsf{Tys} \\
\mathsf{Tms} : \mathsf{Tys} \to \mathsf{Set} & \star : \mathsf{Tms} \diamond \cong \mathbb{1} : \epsilon & (\pi_1, \pi_2) : \mathsf{Tms}\, (\overline{A} \triangleright A) \cong (\overline{a} : \mathsf{Tms}\, \overline{A}) \times \mathsf{Tm}\, (A\, \overline{a})
\end{array}
$$

We can turn the above two isomorphisms into equalities if U in the theory of SOGAT signatures was closed under unit and Σ, and had sort equations at the same time. Note that these are not featured at the same time in the semantics of GATs [43].

Optimizing a Non-Deterministic Abstract Machine with Environments

Małgorzata Biernacka
Institute of Computer Science, University of Wrocław, Poland

Dariusz Biernacki
Institute of Computer Science, University of Wrocław, Poland

Sergueï Lenglet
Université de Lorraine, Nancy, France
Université Sorbonne Paris Nord, France

Alan Schmitt
INRIA, France

Abstract

Non-deterministic abstract machine (NDAM) is a recent implementation model for programming languages where one must choose among several redexes at each reduction step, like process calculi. These machines can be derived from a *zipper semantics*, a mix between structural operational semantics and context-based reduction semantics. Such a machine has been generated also for the λ-calculus without a fixed reduction strategy, i.e., with the full non-deterministic β-reduction.

In that machine, substitution is an external operation that replaces all the occurrences of a variable at once. Implementing substitution with environments is more low-level and more efficient as variables are replaced only when needed. In this paper, we define a NDAM with environments for the λ-calculus without a fixed reduction strategy. We also introduce other optimizations, including a form of refocusing, and we show that we can restrict our optimized NDAM to recover some of the usual λ-calculus machines, e.g., the Krivine Abstract Machine. Most of the improvements we propose in this work could be applied to other NDAMs as well.

2012 ACM Subject Classification Theory of computation → Abstract machines

Keywords and phrases Abstract machine, Explicit substitutions, Refocusing

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.11

Related Version *Extended Version*: `https://inria.hal.science/hal-04568253` [15]

Supplementary Material *Software (Implementation of the Optimized Abstract Machine)*: `https://gitlab.inria.fr/skeletons/lambda-am.git` [13]
archived at `swh:1:dir:ba776d5226cde3b68cc6346fcd3a3495ad4d04e8`

Funding This work is partially funded by the National Science Centre of Poland under grant no. 2019/33/B/ST6/00289, and PHC Polonium.

Acknowledgements We thank the anonymous reviewers for their comments.

1 Introduction

Abstract machines are first-order tail-recursive transition systems for term reduction. Known examples of machines for languages based on the λ-calculus include SECD [39], CEK [32], and the KAM [38]. They serve as an operational semantics [39, 32, 31] or an implementation model [36, 41] of programming languages, but are also relevant in proof theory [38], higher-order model checking [46], or as cost models [10], to name but a few applications.

Abstract machines for the λ-calculus usually implement a *deterministic* strategy which selects a specific redex at each step [27, 31]. In contrast, a Non-Deterministic Abstract Machine (NDAM) [14] may choose any of the available redexes: it explores the term, making

an arbitrary choice when several are possible, e.g., going left or right of an application. If the machine focuses on a subterm which cannot be further reduced, it annotates it and backtracks to make a different choice. Annotations prevent the machine from visiting an already explored subterm.

NDAMs have been originally developed as an implementation model for non-deterministic languages, in particular, for process calculi. However, considering an NDAM for the λ-calculus without a fixed reduction strategy is worthwhile too, at least from the theoretical point of view – one can expect that such an NDAM should subsume various deterministic reduction strategies or abstract machines, and, therefore, play a role of a unifying framework in which one can study and compare them. Moreover, such an abstract machine should be readily adjustable to account for the λ-calculus extended with concurrency primitives.

Defining an NDAM can be hard, so the framework in [14] comes with a sound and complete automatic derivation procedure from another semantics format called *zipper semantics*, which is a middle ground between a Structural Operational Semantics (SOS) [44] and a context-based reduction semantics [32, 26]. Like an SOS, a zipper semantics explores a term with structural rules, but it remembers its position in the term using an evaluation context, i.e., a term with a hole [32]. Zipper semantics and their derived NDAMs have been presented for process calculi [47, 40], but also for the λ-calculus without a fixed reduction strategy.

As a result of the generic derivation procedure, the NDAM for the λ-calculus cannot be perfectly tailored for it, and, moreover, it inherits some of the artifacts from the underlying zipper semantics. In particular, it relies on substitution, an external operation which replaces all the occurrences of a variable at once, traversing the entire term. A more elementary and efficient implementation of β-reduction that is traditionally used in abstract machines uses environments, i.e., delayed substitutions, where a variable is looked up only when needed, and closures are built [39]. The goal of this work is to obtain an environment-based NDAM for the λ-calculus without a fixed reduction strategy.

To this end we first define a zipper semantics with environments for the λ-calculus without a fixed reduction strategy. It is inspired by $\lambda\sigma_{\Uparrow}$, a classic calculus with explicit substitutions [25]. It turns out that we only need to implement a small subset of the rewrite rules of Curien et al.'s calculus in our zipper semantics. Then, the NDAM automatically derived from the zipper semantics itself uses environments.

Unfortunately, the derived NDAM still suffers from performance issues. For example, it is not in refocused form [27, 28]: after a β-reduction, the NDAM reconstructs the whole term before looking for the next redex. The reason is that an NDAM should be able to find any redex, even the ones next to the root. As a compromise between efficiency and completeness, we let the machine reconstruct an arbitrary part of the term from the contraction site. It does not reconstruct the whole term after each β-reduction, but it may still reconstruct enough of it so that it can find β-redexes closer to the root. We also propose other optimizations, in particular related to the way the NDAM manages annotations.

The end product is an optimized NDAM for the λ-calculus with environments. By restricting its behavior, we are then able to express some of the existing λ-calculus machines or strategies, as we expected. For example, if we forbid the exploration step which goes under a λ-abstraction, we restrict ourselves to weak strategies. In addition, if we always go left of an application, we simulate the KAM [38]. We can also encode more complex strategies like full normal order by enforcing priorities on the machine steps of the optimized NDAM. We thus demonstrate that NDAMs can be brought closer to usual, more efficient abstract machines designed for a specific reduction strategy.

Contributions. Our contributions are as follows:

- We define a zipper semantics with environments for the λ-calculus, using a subset of a known calculus with explicit substitutions.
- From the zipper semantics we derive a correct-by-construction non-deterministic abstract machine for the λ-calculus with environments and without a fixed reduction strategy.
- We optimize, in a provably-correct way, the derived NDAM with optimization techniques that we devise to improve the performance of NDAMs in general, which are either generic, like a form of refocusing, or language-specific, like a better management of annotations.
- We provide an OCaml implementation of the optimized machine to experiment with.
- We show that the optimized NDAM we obtain is able to express many of the existing machines designed for a specific reduction strategy.

Structure of the paper. Section 2 recalls the definition of the zipper semantics with substitution, which we characterize with a zipper semantics with environments. In Section 3, we first present the NDAM derived from the environment-based zipper semantics, and then define a more optimized machine. In Section 4, we show how to obtain previously known machines from the optimized NDAM. Section 5 discusses related work and Section 6 concludes the paper. The accompanying report [15] contains the proofs missing from the paper.

2 Zipper Semantics

We present the zipper semantics of the λ-calculus with environments and prove its equivalence with the substitution-based variant.

2.1 Syntax and Substitution-based Zipper Semantics

We present the syntax of the λ-calculus where variables are represented with de Bruijn indices. We let n, m, p range over indices, i.e., natural numbers, and define the syntax of λ-calculus terms, ranged over with t and s, as follows.

$$t, s ::= \lambda.t \mid n \mid t\ s$$

A de Bruijn index counts how many λ-abstractions separate the variable from its binder. For example, using more familiar named variables, $\lambda.0$ stands for $\lambda x.x$, and $\lambda.\lambda.1$ for $\lambda x.\lambda y.x$. We use a de Bruijn representation for the environment-based calculus (Section 2.2), to prevent unwanted name captures when moving an environment inside a λ-abstraction. We write $t\{s/n\}$ for the capture-avoiding substitution of n with s in t, defined as follows, using the auxiliary renaming operation on indices $\langle t\rangle_p^n$ [29].

$$(t_1\ t_2)\{s/n\} \triangleq t_1\{s/n\}\ t_2\{s/n\} \qquad \langle t_1\ t_2\rangle_p^n \triangleq \langle t_1\rangle_p^n\ \langle t_2\rangle_p^n$$

$$(\lambda.t)\{s/n\} \triangleq \lambda.t\{s/n+1\} \qquad \langle \lambda.t\rangle_p^n \triangleq \lambda.\langle t\rangle_{p+1}^n$$

$$m\{s/n\} \triangleq \begin{cases} m-1 & \text{if } m > n \\ \langle s\rangle_0^n & \text{if } m = n \\ m & \text{if } m < n \end{cases} \qquad \langle m\rangle_p^n \triangleq \begin{cases} m+n & \text{if } m \geq p \\ m & \text{if } m < p \end{cases}$$

The goal of zipper semantics is to go through a term looking for a redex, using an evaluation context to remember the current focus in the term. We represent evaluation contexts $\mathbb{E}$, $\mathbb{F}$ as stacks of elementary contexts $\mathfrak{F}$, called *frames*.

$$\mathbb{E}, \mathbb{F} ::= \mathfrak{F} :: \mathbb{E} \mid \blacksquare \qquad \mathfrak{F} ::= \square\ t \mid t\ \square \mid \lambda$$

$$\textsf{appL}\ \frac{t \xrightarrow{\square\, s :: \mathbb{E}}_{\mathsf{ev}} t'}{t\ s \xrightarrow{\mathbb{E}}_{\mathsf{ev}} t'} \qquad \textsf{appR}\ \frac{s \xrightarrow{t\, \square :: \mathbb{E}}_{\mathsf{ev}} s'}{t\ s \xrightarrow{\mathbb{E}}_{\mathsf{ev}} s'} \qquad \textsf{lam}\ \frac{t \xrightarrow{\lambda :: \mathbb{E}}_{\mathsf{ev}} t'}{\lambda.t \xrightarrow{\mathbb{E}}_{\mathsf{ev}} t'} \qquad \textsf{lam}\beta\ \frac{}{\lambda.t \xrightarrow{\square\, s :: \mathbb{E}}_{\mathsf{ev}} \mathbb{E}\langle t\{s/0\}\rangle}$$

Figure 1 Zipper semantics of the substitution-based lambda-calculus.

In defining the zipper semantics, it is convenient to interpret contexts *inside-out*: the topmost frame represents the innermost construct. We define a plugging operation for frames $\mathfrak{F}\langle t\rangle$ and contexts $\mathbb{E}\langle t\rangle$ as follows.

$$(\square\ s)\langle t\rangle \triangleq t\ s \qquad (s\ \square)\langle t\rangle \triangleq s\ t \qquad \lambda\langle t\rangle \triangleq \lambda.t$$
$$\blacksquare\langle t\rangle \triangleq t \qquad (\mathfrak{F} :: \mathbb{E})\langle t\rangle \triangleq \mathbb{E}\langle \mathfrak{F}\langle t\rangle\rangle$$

The definition of the zipper semantics is in Figure 1. Informally, $t \xrightarrow{\mathbb{E}}_{\mathsf{ev}} t'$ means that $\mathbb{E}\langle t\rangle$ reduces to t'. The rules appL, appR, and lam are going through the term looking for a redex. For example in appL, $t\ s$ reduces to t' in $\mathbb{E}$ if t reduces to t' in $\square\ s :: \mathbb{E}$. We see that the focus changes from $t\ s$ to t, resulting in pushing $\square\ s$ on top of $\mathbb{E}$. When we focus on a λ-abstraction inside a context $\square\ s :: \mathbb{E}$, we recognize a redex, and we can β-reduce with the axiom lamβ.

A reduction step is defined as a transition $t \xrightarrow{\blacksquare}_{\mathsf{ev}} t'$. The semantics of Figure 1 is a slight optimization of the original zipper semantics for the λ-calculus [14], where the axiom lamβ is split in two rules. It is however straightforward to prove that the two zipper semantics coincide, and therefore our semantics also characterizes the usual context-based reduction semantics of the λ-calculus [14].

2.2 Environment-based Zipper Semantics

Our goal is to replace substitution with environments, while still being able to reduce an arbitrary β-redex, including under a λ-abstraction. To this end, we take inspiration from $\lambda\sigma_{\Uparrow}$ [25], a calculus with explicit substitutions and strong reduction. The syntax of *environments* ϵ, ϕ and of *closures* c, d are as follows.

$$\epsilon, \phi ::= \mathtt{id} \mid \uparrow \mid l \cdot \epsilon \mid \epsilon \circ \phi \mid \Uparrow\epsilon \qquad c, d ::= l[\epsilon] \mid \lambda.c \mid c\ d \qquad l, k ::= t \mid c$$

Closures extend terms by pairing them with an environment $t[\epsilon]$. Environments are like explicit substitutions in $\lambda\sigma_{\Uparrow}$. An environment $l \cdot \epsilon$ maps 0 to l and the variables $n \geq 1$ are mapped by ϵ. The identity environment $\mathtt{id}$ maps variables to themselves. The composition $\epsilon \circ \phi$ is such that ϵ is applied first, and then ϕ.

Whenever an environment ϵ is applied to the body of a λ-abstraction $\lambda.l$, it is *lifted* $\Uparrow\epsilon$ to avoid index clashes. Informally, ϵ should not be applied to 0 in l, as 0 is bound by the λ, and the occurrences of 0 in the image of ϵ should not be captured by that λ. To avoid this, we *shift* ($\uparrow$) the result of the application of ϵ. To summarize, $\Uparrow\epsilon$ acts like $0 \cdot \epsilon \circ \uparrow$: the variable 0 is mapped to 0, and any $n \geq 1$ is mapped by ϵ and the result is then shifted. This reminds that the lift construct is not mandatory in calculi with explicit substitutions, as its behavior can be expressed with the other constructs [1, 25]: we use it because it is more convenient in proofs, in particular to prove termination results.

The zipper semantics for the environment-based λ-calculus is defined in Figure 2. It takes as source a term or a closure l, but always results in a closure c. As in the substitution-based calculus, we use a context to remember the current position when exploring l. With

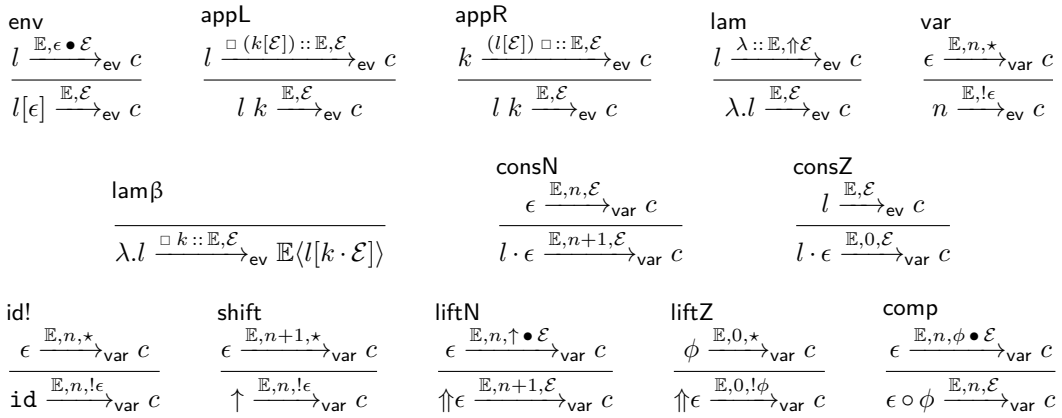


Figure 2 Zipper semantics for the environment-based λ-calculus.

environments, context frames are built from terms or closures: $\mathfrak{F} ::= \square\, l \mid l\, \square \mid \lambda$. An evaluation context $\mathbb{E}$, $\mathbb{F}$ is still a stack of frames interpreted inside-out, and we plug terms or closures inside a frame or a context. The plugging operation is defined as in Section 2.1.

The zipper semantics uses an extra argument to record what the current environment is while exploring l. To this end, we define an option type $\mathcal{E}$, $\mathcal{F}$, which we call *local environment*, signalling whether the environment is defined or not.

$$\mathcal{E}, \mathcal{F} ::= \star \mid !\epsilon$$

When $\mathcal{E} = \star$, the local environment is not (yet) defined. Although $\star$ could be seen as $\mathtt{id}$, we distinguish the two to make explicit the absence or presence of an environment. We extend the following constructs to local environments and overload some of the symbols. The rationale behind these definitions is explained when we detail the rules of the zipper semantics.

$$l[\star] \triangleq l \qquad l \cdot \star \triangleq l \cdot \mathtt{id} \qquad \epsilon \bullet \star \triangleq !\epsilon \qquad \Uparrow\star \triangleq \star$$
$$l[!\epsilon] \triangleq l[\epsilon] \qquad l \cdot !\epsilon \triangleq l \cdot \epsilon \qquad \epsilon_1 \bullet !\epsilon_2 \triangleq !\epsilon_1 \circ \epsilon_2 \qquad \Uparrow!\epsilon \triangleq !\Uparrow\epsilon$$

Zipper semantics transitions are of two kinds: $l \xrightarrow{\mathbb{E},\mathcal{E}}_{\mathsf{ev}} c$ means that l with local environment $\mathcal{E}$ inside $\mathbb{E}$ is reducing to c, i.e., $\mathbb{E}\langle l[\mathcal{E}]\rangle$ is β-reducing to c. The transition $\epsilon \xrightarrow{\mathbb{E},n,\mathcal{E}}_{\mathsf{var}} c$ means that solving n in the environment $\epsilon \bullet \mathcal{E}$ results in a term or closure which itself reduces within $\mathbb{E}$ to c: $\mathbb{E}\langle n[\epsilon \bullet \mathcal{E}]\rangle$ is β-reducing to c. The rules defining the semantics implement some of the rewriting rules of $\lambda\sigma_{\Uparrow}$.

The transition $l \xrightarrow{\mathbb{E},\mathcal{E}}_{\mathsf{ev}} c$ is going through l until it finds a β-redex, keeping track of the current environment $\mathcal{E}$ while doing so. We start with an empty context and an undefined environment $l \xrightarrow{\blacksquare,\star}_{\mathsf{ev}} c$. When we focus on a construct $l[\epsilon]$ (rule env), we set the local environment to $!\epsilon$ if it was undefined; otherwise, we compose it with ϵ. The definition of $\bullet$ reflects these two cases.

In the application case (rules appL and appR), we pack the local environment with the term or closure pushed on the context. In the body of a λ-abstraction, the local environment becomes $\Uparrow\mathcal{E}$ (rule lam): if it is undefined ($\mathcal{E} = \star$), it remains so in the body of the abstraction, which explains why $\Uparrow\star$ is defined as $\star$. Otherwise, $\mathcal{E} = !\epsilon$, and ϵ is lifted. For a variable n

$$
\begin{array}{lll}
[\![l[\epsilon_1]]\!](\epsilon_2) \triangleq [\![[\![l]\!](\epsilon_1)]\!](\epsilon_2) & [\![0]\!](l\cdot\epsilon) \triangleq [\![l]\!] & [\![n+1]\!](l\cdot\epsilon) \triangleq [\![n]\!](\epsilon) \\
[\![l_1\ l_2]\!](\epsilon) \triangleq [\![l_1]\!](\epsilon)\ [\![l_2]\!](\epsilon) & [\![0]\!](\Uparrow\epsilon) \triangleq 0 & [\![n+1]\!](\Uparrow\epsilon) \triangleq [\![n]\!](\epsilon\circ\uparrow) \\
[\![\lambda.l]\!](\epsilon) \triangleq \lambda.[\![l]\!](\Uparrow\epsilon) & [\![n]\!](\mathtt{id}) \triangleq n & \\
[\![n]\!](\epsilon_1\circ\epsilon_2) \triangleq [\![[\![n]\!](\epsilon_1)]\!](\epsilon_2) & [\![n]\!](\uparrow) \triangleq n+1 &
\end{array}
$$

Figure 3 Translation from closures to terms.

(rule var), $\mathcal{E}$ should be defined as $!\epsilon$ for some ϵ, so that we can resolve n in ϵ using the var transition, explained below. Finally, if we focus on a λ-abstraction $\lambda.l$ in the appropriate context $\square\ k :: \mathbb{E}$, we can β-reduce, which consists in pushing k on top of $\mathcal{E}$; if $\mathcal{E} = \star$, then this operation creates the environment $k\cdot\mathtt{id}$ (rule lamβ).

The transition $\epsilon \xrightarrow{\mathbb{E},n,\mathcal{E}}_{\mathsf{var}} c$ resolves n in ϵ, using $\mathcal{E}$ to handle compositions. Initially in rule var, $\mathcal{E}$ is set to $\star$. If $\epsilon = \epsilon_1\circ\epsilon_2$, then we focus on ϵ_1 and remember ϵ_2 in $\mathcal{E}$ (rule comp). More generally, the transition $\epsilon \xrightarrow{\mathbb{E},n,\star}_{\mathsf{var}} c$ is solving n in ϵ, while $\epsilon \xrightarrow{\mathbb{E},n,!\phi}_{\mathsf{var}} c$ is solving n in $\epsilon\circ\phi$. For example, the rule shift says that resolving n in $\uparrow\circ\phi$ consists in resolving $n+1$ in ϕ. Similarly, solving n in $\mathtt{id}\circ\phi$ requires solving n in ϕ (rule id!).

As explained before, $\Uparrow\epsilon$ maps 0 to 0 (rule liftZ) and any $n \geq 1$ is solved in ϵ and then shifted (rule liftN). The rules consN and consZ deal with environments $l\cdot\epsilon$ as expected: if $n \geq 1$, then we look for $n-1$ in ϵ. Otherwise, l is the expected result, and we switch to a ev transition to continue the search for a redex.

The var transition is not defined when $\mathcal{E} = \star$ and the environment is $\mathtt{id}$, $\uparrow$, or $\Uparrow\epsilon$ with $n = 0$, i.e., in the cases producing a variable (respectively n, $n+1$, and 0). The zipper semantics is solving a variable to find a β-redex, and turning a variable into another variable does not help in that regard.

2.3 Properties of the Environment-based Zipper Semantics

We state the main properties of the environment-based zipper semantics. We first show it corresponds to the substitution-based semantics. To this end, in Figure 3 we define a translation $[\![l]\!](\epsilon)$ from closures to terms: $[\![l]\!](\epsilon) = t$ means that the term or closure l under environment ϵ translates into the term t. The translation is defined first by case analysis on the term or closure l, to push the environment ϵ to the leaves of l. At the level of variables, it then computes how an index n is translated by case analysis on the environment.

We write $[\![l]\!]$ as a shorthand for $[\![l]\!](\mathtt{id})$. The equations defining the translation contain the rewrite rules we need from the ones defining $\lambda\sigma_{\Uparrow}$. We can derive some of the remaining rules of $\lambda\sigma_{\Uparrow}$ as properties of the translation: for example, we can show that $[\![l]\!](\Uparrow\mathtt{id}) = [\![l]\!](\mathtt{id})$ [15, Lemma 7] or that $[\![l]\!](\epsilon\circ\phi) = [\![[\![l]\!](\epsilon)]\!](\phi)$ [15, Lemma 12]. The proofs are usually by induction on the size of pairs (l,ϵ), strictly decreasing between the left-hand side and right-hand side of each equation defining the translation [15, Appendix A]. The size is the same as in the proof of termination of rewriting in $\lambda\sigma_{\Uparrow}$ [25, Proposition 4.2], and relies on the fact that we use lift; the proof would be more complex without lift [1].

The main result relates β-redexes on both sides.

▶ **Lemma 1.** *For all l, k, ϵ, $[\![l[k\cdot\epsilon]]\!] = [\![l]\!](\Uparrow\epsilon)\{[\![k]\!]/0\}$.*

Curien et al. prove a similar result when relating $\lambda\sigma_{\Uparrow}$ to the plain λ-calculus [25, Proposition 4.11]: rephrased in our setting, they prove that given two terms t and s, $[\![t[s \cdot \mathtt{id}]]\!] = t\{s/0\}$. We generalize their result to any closures l, k and any environment ϵ. With this result, we can relate the two zipper semantics.

▶ **Theorem 2.** *If* $l \xrightarrow{\blacksquare,\star}_{\mathsf{ev}} c$, *then* $[\![l]\!] \xrightarrow{\blacksquare}_{\mathsf{ev}} [\![c]\!]$.
For all $t \xrightarrow{\blacksquare}_{\mathsf{ev}} s$, *there exists* c *such that* $t \xrightarrow{\blacksquare,\star}_{\mathsf{ev}} c$ *and* $[\![c]\!] = s$.

The environment-based semantics is equivalent to the substitution-based one, which itself characterizes a context-based semantics of the λ-calculus [14].

The environment-based zipper semantics also satisfies the properties required so it can be derived into an NDAM [14]. In particular, it must be *machine constructive*: in each rule, it is possible to construct the terms in the premise from those of the conclusion. The semantics must also be *terminating*, ensuring that the search for a redex in the abstract machine does not loop infinitely. To prove it, it is enough to exhibit a strictly decreasing size of transitions such that the size of the premise of each inductive rule is smaller than that of its conclusion. The definition of such a size is already necessary to prove Theorem 2 [15, Appendix A]. We point out that the size is strictly decreasing because we distinguish $\star$ from $\mathtt{id}$, which allows us to give different sizes for both. Replacing $\star$ with $\mathtt{id}$ would make the zipper rule $\mathsf{id!}$ in Figure 2 not terminating.

The original theory [14] requires the zipper semantics to be *reversible*, which is the reverse of being machine constructive: for each rule, the terms in the conclusion can be deduced from those in the premise. This property makes it easier to design the backtracking mechanism of the derived machine. The zipper semantics of Figure 2 is not reversible: in rules consN and consZ, respectively l and ϵ in the conclusion do not occur in the premise. We also need to inverse the operator $\bullet$ in rules env, liftN, and comp. We show in Section 3.1 that we only need a small change in the derivation procedure to allow backtracking for these rules. However, we argue that the derivation procedure could easily be adapted to account for non-reversible zipper semantics in many cases.

3 Non-Deterministic Abstract Machines

We discuss the shortcomings of the NDAM derived from the environment-based semantics, and present an optimized abstract machine dealing with these issues.

3.1 Derived Abstract Machine

A zipper semantics is not yet an implementation, as it does not explain what to do when several rules can be applied during the search for a redex, like in the application or λ-abstraction cases. The automatically derived abstract machine (DAM in short) augments the zipper semantics with a backtracking mechanism and annotations to handle these cases.

Roughly, each zipper rule becomes a *forward* step, which focuses on a subterm. When several rules can be applied, the machine chooses arbitrarily one of them, e.g., going left or right of an application. Because of this non-determinism, it can make a wrong choice and focus on a subterm which cannot reduce. In such a case, the machine *annotates* the subterm as a normal form, and then switches to a *backward* mode, in which it undoes enough steps to return to the last configuration it made a choice, to make a different one if possible. The forward steps may focus only on subterms without annotations, ensuring that the machine cannot make the same (wrong) choice twice. The DAM relies on a stack π to remember in which order the forward steps have been applied. The backward mode then uses this stack to undo the steps in the reverse order.

Formally, we let Σ, Δ range over annotations, and we extend the syntax of terms, closures, and environments with annotations.

$$\Sigma, \Delta ::= \emptyset \mid \mathsf{ev} \mid \mathsf{var} \qquad t, s ::= n^{\Sigma} \mid \lambda^{\Sigma}.t \mid t \mathbin{@^{\Sigma}} s$$

$$\epsilon, \phi ::= \mathtt{id}^{\Sigma} \mid \uparrow^{\Sigma} \mid l \overset{\Sigma}{\cdot} \epsilon \mid \epsilon \overset{\Sigma}{\circ} \phi \mid \Uparrow^{\Sigma} \epsilon \qquad c, d ::= l[\epsilon]^{\Sigma} \mid \lambda^{\Sigma}.c \mid c \mathbin{@^{\Sigma}} d$$

We use l and k to range over (annotated) terms and closures. The machine annotates a term or closure with ev and an environment with var. We write $\mathsf{an}(l)$ or $\mathsf{an}(\epsilon)$ for the topmost annotation of l or ϵ: for instance $\mathsf{an}(l \mathbin{@^{\Sigma}} k) = \Sigma$. We write $l^{+\mathsf{ev}}$ and $\epsilon^{+\mathsf{var}}$ for the operation of annotating the topmost construct of l or ϵ: for example, $(l \mathbin{@^{\Sigma}} k)^{+\mathsf{ev}} = l \mathbin{@^{\mathsf{ev}}} k$ independently of Σ. Contexts in machine configurations are built of annotated terms, closures, and environments.

The syntax of the stack π is of the shape $\pi ::= \rho(\tilde{e}) :: \pi \mid \blacktriangle$ where $\blacktriangle$ represents the empty stack, ρ ranges over the names of the zipper rules in Figure 2, and $\tilde{e}$ represents information stored in the stack, which might be needed to make backtracking possible. In most cases, $\tilde{e}$ is empty.

The DAM is defined in Figure 4; we discuss some of the steps here. Machine configurations $\langle l \mid \mathbb{E}, \mathcal{E}, \pi \rangle_{\mathsf{ev}}$ and $\langle \epsilon \mid \mathbb{E}, n, \mathcal{E}, \pi \rangle_{\mathsf{var}}$ reflect the transitions of the zipper semantics. Each zipper rule is turned into a forward step, where the conclusion and premise of the rule are respectively the source and target of the step. The stack π also records the rule being applied. For example, the rules appL, lam and consN translate as steps D1, D3, and D14, respectively. Each forward step has a side condition which checks that the new subterm under focus is not annotated, thus ensuring it has not already been visited.

If no forward step applies to the current term, then a switching step annotates it and changes to the backward mode bev or bvar. The configurations $\langle \pi \mid l, \mathbb{E}, \mathcal{E} \rangle_{\mathsf{bev}}$ and $\langle \pi \mid \epsilon, \mathbb{E}, n, \mathcal{E} \rangle_{\mathsf{var}}$ then discriminate on π to undo the last applied rule. Each forward step has a mirroring backward step to undo it. The switching steps are steps D7 and D21 for the ev and var modes respectively, and the backward steps corresponding to appL, lam and consN are steps D8, D11, and D22, respectively.

Finally, when the machine finds a redex, it applies the axiom $\mathsf{lam}\beta$ (step D6). The machine constructs the result of the β-reduction plugged inside $\mathbb{E}$. After reduction, some annotations may no longer be valid, so to be safe, the DAM erases all of them: the operation $|l|$ sets every annotation in l to $\emptyset$.

Reversibility. In the the original derivation procedure [14], the role of π is mainly to record which rules are being applied, to know in which order to undo them if needed. We go beyond that design in this paper, and use π to also store some information to be able to backtrack even in the case of non-reversible zipper rules. We argue that what we do is generic enough to be added to the original derivation procedure.

The rules consN, consZ, and liftZ are not reversible because some entities of the conclusion of the rule do not occur in its premise. It is easy to automatically collect such missing entities, and record them in the stack π. For example, the steps D14 and D15 – corresponding to consN and consZ – remember in π the part of the environment which is discarded, either its head or its tail.

A rule may be not reversible also because it relies on an external operator which is not injective, like $\bullet$. In that case, we need to provide extra operations and remember some information in the stack to be able to invert it. By its definition, $\epsilon \bullet \mathcal{E} = !\epsilon_1 \circ \epsilon_2$ because either $\epsilon = \epsilon_1$ and $\mathcal{E} = !\epsilon_2$, or $\epsilon = \epsilon_1 \circ \epsilon_2$ and $\mathcal{E} = \star$. To distinguish between the two possibilities, we remember in the stack π a boolean b defined by an operation $\mathcal{E} \overset{?}{=} \star$ which evaluates to $\top$

$$\langle l_1 \mathbin{@^{\emptyset}} l_2 \mid \mathbb{E}, \mathcal{E}, \pi\rangle_{\mathsf{ev}} \to \langle l_1 \mid \Box\,(l_2[\mathcal{E}]) :: \mathbb{E}, \mathcal{E}, \mathsf{appl} :: \pi\rangle_{\mathsf{ev}} \qquad \text{if } \mathsf{an}(l_1) = \emptyset \qquad \text{(D1)}$$

$$\langle l_1 \mathbin{@^{\emptyset}} l_2 \mid \mathbb{E}, \mathcal{E}, \pi\rangle_{\mathsf{ev}} \to \langle l_2 \mid (l_1[\mathcal{E}])\,\Box :: \mathbb{E}, \mathcal{E}, \mathsf{appr} :: \pi\rangle_{\mathsf{ev}} \qquad \text{if } \mathsf{an}(l_2) = \emptyset \qquad \text{(D2)}$$

$$\langle \lambda^{\emptyset}.l \mid \mathbb{E}, \mathcal{E}, \pi\rangle_{\mathsf{ev}} \to \langle l \mid \lambda :: \mathbb{E}, \Uparrow\mathcal{E}, \mathsf{lam} :: \pi\rangle_{\mathsf{ev}} \qquad \text{if } \mathsf{an}(l) = \emptyset \qquad \text{(D3)}$$

$$\langle l[\epsilon]^{\emptyset} \mid \mathbb{E}, \mathcal{E}, \pi\rangle_{\mathsf{ev}} \to \langle l \mid \mathbb{E}, \epsilon \bullet \mathcal{E}, \mathsf{env}(\mathcal{E} \stackrel{?}{=} \star) :: \pi\rangle_{\mathsf{ev}} \qquad \text{if } \mathsf{an}(l) = \emptyset \qquad \text{(D4)}$$

$$\langle n^{\emptyset} \mid \mathbb{E}, !\epsilon, \pi\rangle_{\mathsf{ev}} \to \langle \epsilon \mid \mathbb{E}, n, \star, \mathsf{var} :: \pi\rangle_{\mathsf{var}} \qquad \text{if } \mathsf{an}(\epsilon) = \emptyset \qquad \text{(D5)}$$

$$\langle \lambda^{\emptyset}.l \mid \Box\, k :: \mathbb{E}, \mathcal{E}, \pi\rangle_{\mathsf{ev}} \to \langle |\mathbb{E}\langle l[k \mathbin{\cdot^{\emptyset}} \mathcal{E}]^{\emptyset}\rangle| \mid \blacksquare, \star, \blacktriangle\rangle_{\mathsf{ev}} \qquad \text{(D6)}$$

$$\langle l \mid \mathbb{E}, \mathcal{E}, \pi\rangle_{\mathsf{ev}} \to \langle \pi \mid l^{+\mathsf{ev}}, \mathbb{E}, \mathcal{E}\rangle_{\mathsf{bev}} \qquad \text{otherwise} \qquad \text{(D7)}$$

$$\langle \mathsf{appl} :: \pi \mid l_1, \Box\,(l_2[\mathcal{E}]) :: \mathbb{E}, \mathcal{E}\rangle_{\mathsf{bev}} \to \langle l_1 \mathbin{@^{\emptyset}} l_2 \mid \mathbb{E}, \mathcal{E}, \pi\rangle_{\mathsf{ev}} \qquad \text{(D8)}$$

$$\langle \mathsf{appr} :: \pi \mid l_2, (l_1[\mathcal{E}])\,\Box :: \mathbb{E}, \mathcal{E}\rangle_{\mathsf{bev}} \to \langle l_1 \mathbin{@^{\emptyset}} l_2 \mid \mathbb{E}, \mathcal{E}, \pi\rangle_{\mathsf{ev}} \qquad \text{(D9)}$$

$$\langle \mathsf{env}(b) :: \pi \mid l, \mathbb{E}, \mathcal{E}'\rangle_{\mathsf{bev}} \to \langle l[\epsilon]^{\emptyset} \mid \mathbb{E}, \mathcal{E}, \pi\rangle_{\mathsf{ev}} \qquad \text{if } \bullet^{-1}(b, \mathcal{E}') = (\epsilon, \mathcal{E}) \qquad \text{(D10)}$$

$$\langle \mathsf{lam} :: \pi \mid l, \lambda :: \mathbb{E}, \Uparrow\mathcal{E}\rangle_{\mathsf{bev}} \to \langle \lambda^{\emptyset}.l \mid \mathbb{E}, \mathcal{E}, \pi\rangle_{\mathsf{ev}} \qquad \text{(D11)}$$

$$\langle \mathsf{var} :: \pi \mid \epsilon, \mathbb{E}, n, \star\rangle_{\mathsf{bvar}} \to \langle n^{\emptyset} \mid \mathbb{E}, !\epsilon, \pi\rangle_{\mathsf{ev}} \qquad \text{(D12)}$$

$$\langle \blacktriangle \mid l, \blacksquare, \mathcal{E}\rangle_{\mathsf{bev}} \to \langle l[\mathcal{E}]\rangle_{\mathsf{nf}} \qquad \text{(D13)}$$

$$\langle l \mathbin{\cdot^{\emptyset}} \epsilon \mid \mathbb{E}, n+1, \mathcal{E}, \pi\rangle_{\mathsf{var}} \to \langle \epsilon \mid \mathbb{E}, n, \mathcal{E}, \mathsf{consN}(l) :: \pi\rangle_{\mathsf{var}} \qquad \text{if } \mathsf{an}(\epsilon) = \emptyset \qquad \text{(D14)}$$

$$\langle l \mathbin{\cdot^{\emptyset}} \epsilon \mid \mathbb{E}, 0, \mathcal{E}, \pi\rangle_{\mathsf{var}} \to \langle l \mid \mathbb{E}, \mathcal{E}, \mathsf{cons0}(\epsilon) :: \pi\rangle_{\mathsf{ev}} \qquad \text{if } \mathsf{an}(l) = \emptyset \qquad \text{(D15)}$$

$$\langle \mathtt{id}^{\emptyset} \mid \mathbb{E}, n, !\epsilon, \pi\rangle_{\mathsf{var}} \to \langle \epsilon \mid \mathbb{E}, n, \star, \mathsf{id}! :: \pi\rangle_{\mathsf{var}} \qquad \text{if } \mathsf{an}(\epsilon) = \emptyset \qquad \text{(D16)}$$

$$\langle \uparrow^{\emptyset} \mid \mathbb{E}, n, !\epsilon, \pi\rangle_{\mathsf{var}} \to \langle \epsilon \mid \mathbb{E}, n+1, \star, \mathsf{shift} :: \pi\rangle_{\mathsf{var}} \qquad \text{if } \mathsf{an}(\epsilon) = \emptyset \qquad \text{(D17)}$$

$$\langle \Uparrow^{\emptyset}\epsilon \mid \mathbb{E}, n+1, \mathcal{E}, \pi\rangle_{\mathsf{var}} \to \langle \epsilon \mid \mathbb{E}, n, \uparrow^{\emptyset} \bullet \mathcal{E}, \mathsf{liftN}(\mathcal{E} \stackrel{?}{=} \star) :: \pi\rangle_{\mathsf{var}} \qquad \text{if } \mathsf{an}(\epsilon) = \emptyset \qquad \text{(D18)}$$

$$\langle \Uparrow^{\emptyset}\epsilon \mid \mathbb{E}, 0, !\phi, \pi\rangle_{\mathsf{var}} \to \langle \phi \mid \mathbb{E}, 0, \star, \mathsf{lift0}(\epsilon) :: \pi\rangle_{\mathsf{var}} \qquad \text{if } \mathsf{an}(\epsilon_2) = \emptyset \qquad \text{(D19)}$$

$$\langle \epsilon_1 \mathbin{\circ^{\emptyset}} \epsilon_2 \mid \mathbb{E}, n, \mathcal{E}, \pi\rangle_{\mathsf{var}} \to \langle \epsilon_1 \mid \mathbb{E}, n, \epsilon_2 \bullet \mathcal{E}, \mathsf{comp}(\mathcal{E} \stackrel{?}{=} \star) :: \pi\rangle_{\mathsf{var}} \qquad \text{if } \mathsf{an}(\epsilon_1) = \emptyset \qquad \text{(D20)}$$

$$\langle \epsilon \mid \mathbb{E}, n, \mathcal{E}, \pi\rangle_{\mathsf{var}} \to \langle \pi \mid \epsilon^{+\mathsf{var}}, \mathbb{E}, n, \mathcal{E}\rangle_{\mathsf{bvar}} \qquad \text{otherwise} \qquad \text{(D21)}$$

$$\langle \mathsf{consN}(l) :: \pi \mid \epsilon, \mathbb{E}, n, \mathcal{E}\rangle_{\mathsf{bvar}} \to \langle l \mathbin{\cdot^{\emptyset}} \epsilon \mid \mathbb{E}, n+1, \mathcal{E}, \pi\rangle_{\mathsf{var}} \qquad \text{(D22)}$$

$$\langle \mathsf{cons0}(\epsilon) :: \pi \mid l, \mathbb{E}, \mathcal{E}\rangle_{\mathsf{bev}} \to \langle l \mathbin{\cdot^{\emptyset}} \epsilon \mid \mathbb{E}, 0, \mathcal{E}, \pi\rangle_{\mathsf{var}} \qquad \text{(D23)}$$

$$\langle \mathsf{id}! :: \pi \mid \epsilon, \mathbb{E}, n, \star\rangle_{\mathsf{bvar}} \to \langle \mathtt{id}^{\emptyset} \mid \mathbb{E}, n, !\epsilon, \pi\rangle_{\mathsf{var}} \qquad \text{(D24)}$$

$$\langle \mathsf{shift} :: \pi \mid \epsilon, \mathbb{E}, n+1, \star, \rangle_{\mathsf{bvar}} \to \langle \uparrow^{\emptyset} \mid \mathbb{E}, n, !\epsilon, \pi\rangle_{\mathsf{var}} \qquad \text{(D25)}$$

$$\langle \mathsf{liftN}(b) :: \pi \mid \epsilon, \mathbb{E}, n, \mathcal{E}'\rangle_{\mathsf{bvar}} \to \langle \Uparrow^{\emptyset}\epsilon \mid \mathbb{E}, n+1, \mathcal{E}, \pi\rangle_{\mathsf{var}} \qquad \text{if } \bullet^{-1}(b, \mathcal{E}') = (\uparrow^{\emptyset}, \mathcal{E}) \qquad \text{(D26)}$$

$$\langle \mathsf{lift0}(\epsilon_1) :: \pi \mid \epsilon_2, \mathbb{E}, 0, \star\rangle_{\mathsf{bvar}} \to \langle \Uparrow^{\emptyset}\epsilon_1 \mid \mathbb{E}, 0, !\epsilon_2, \pi\rangle_{\mathsf{var}} \qquad \text{(D27)}$$

$$\langle \mathsf{comp}(b) :: \pi \mid \epsilon_1, \mathbb{E}, n, \mathcal{E}'\rangle_{\mathsf{bvar}} \to \langle \epsilon_1 \mathbin{\circ^{\emptyset}} \epsilon_2 \mid \mathbb{E}, n, \mathcal{E}, \pi\rangle_{\mathsf{var}} \qquad \text{if } \bullet^{-1}(b, \mathcal{E}') = (\epsilon_2, \mathcal{E}) \qquad \text{(D28)}$$

Figure 4 Derived Abstract Machine.

$$\langle l_1 \mathbin{@^\Sigma} l_2 \mid \mathbb{E}, \mathcal{E}\rangle_{\mathsf{ev}} \to \langle l_1 \mid \square\ (l_2[\mathcal{E}]) :: \mathbb{E}, \mathcal{E}\rangle_{\mathsf{ev}} \quad \text{if } \mathcal{E}, \top \not\Vdash l_1 \tag{O1}$$
$$\langle l_1 \mathbin{@^\Sigma} l_2 \mid \mathbb{E}, \mathcal{E}\rangle_{\mathsf{ev}} \to \langle l_2 \mid (l_1[\mathcal{E}])\ \square :: \mathbb{E}, \mathcal{E}\rangle_{\mathsf{ev}} \quad \text{if } \mathcal{E}, \bot \not\Vdash l_2 \tag{O2}$$
$$\langle \lambda^\Sigma.l \mid \mathbb{E}, \mathcal{E}\rangle_{\mathsf{ev}} \to \langle l \mid \lambda :: \mathbb{E}, \Uparrow\mathcal{E}\rangle_{\mathsf{ev}} \quad \text{if } \Uparrow\mathcal{E}, \bot \not\Vdash l \tag{O3}$$
$$\langle l[\epsilon]^\Sigma \mid \mathbb{E}, \mathcal{E}\rangle_{\mathsf{ev}} \to \langle l \mid \mathbb{E}, \epsilon \bullet \mathcal{E}\rangle_{\mathsf{ev}} \tag{O4}$$
$$\langle n \mid \mathbb{E}, !\epsilon\rangle_{\mathsf{ev}} \to \langle \epsilon \mid \mathbb{E}, n, \star, (n, \epsilon)\rangle_{\mathsf{var}} \tag{O5}$$
$$\langle \lambda^\Sigma.l \mid \square\ k :: \mathbb{E}, \mathcal{E}\rangle_{\mathsf{ev}} \to \langle \mathbb{E} \mid l[k \cdot \mathcal{E}]^\emptyset\rangle_{\mathsf{rec}} \tag{O6}$$
$$\langle n \mid \mathbb{E}, \star\rangle_{\mathsf{ev}} \to \langle \mathbb{E} \mid n[\mathtt{id}]^{\mathsf{ev}}\rangle_{\mathsf{bev}} \tag{O7}$$
$$\langle c \mid \mathbb{E}, \star\rangle_{\mathsf{ev}} \to \langle \mathbb{E} \mid c^{+\mathsf{ev}}\rangle_{\mathsf{bev}} \quad \text{otherwise} \tag{O8}$$

$$\langle \square\ l :: \mathbb{E} \mid c\rangle_{\mathsf{bev}} \to \langle c \mathbin{@^\emptyset} l \mid \mathbb{E}, \star\rangle_{\mathsf{ev}} \tag{O9}$$
$$\langle l\ \square :: \mathbb{E} \mid c\rangle_{\mathsf{bev}} \to \langle l \mathbin{@^\emptyset} c \mid \mathbb{E}, \star\rangle_{\mathsf{ev}} \tag{O10}$$
$$\langle \lambda :: \mathbb{E} \mid c\rangle_{\mathsf{bev}} \to \langle \lambda^\emptyset.c \mid \mathbb{E}, \star\rangle_{\mathsf{ev}} \tag{O11}$$
$$\langle \blacksquare \mid c\rangle_{\mathsf{bev}} \to \langle c\rangle_{\mathsf{nf}} \tag{O12}$$

$$\langle l \cdot \epsilon \mid \mathbb{E}, n+1, \mathcal{E}, P\rangle_{\mathsf{var}} \to \langle \epsilon \mid \mathbb{E}, n, \mathcal{E}, P\rangle_{\mathsf{var}} \tag{O13}$$
$$\langle l \cdot \epsilon \mid \mathbb{E}, 0, \mathcal{E}, P\rangle_{\mathsf{var}} \to \langle l \mid \mathbb{E}, \mathcal{E}\rangle_{\mathsf{ev}} \quad \text{if } \mathcal{E}, \mathsf{infun}(\mathbb{E}) \not\Vdash l \tag{O14}$$
$$\langle \mathtt{id} \mid \mathbb{E}, n, !\epsilon, P\rangle_{\mathsf{var}} \to \langle \epsilon \mid \mathbb{E}, n, \star, P\rangle_{\mathsf{var}} \tag{O15}$$
$$\langle \uparrow \mid \mathbb{E}, n, !\epsilon, P\rangle_{\mathsf{var}} \to \langle \epsilon \mid \mathbb{E}, n+1, \star, P\rangle_{\mathsf{var}} \tag{O16}$$
$$\langle \Uparrow\epsilon \mid \mathbb{E}, n+1, \mathcal{E}, P\rangle_{\mathsf{var}} \to \langle \epsilon \mid \mathbb{E}, n, \uparrow \bullet \mathcal{E}, P\rangle_{\mathsf{var}} \tag{O17}$$
$$\langle \Uparrow\epsilon \mid \mathbb{E}, 0, !\phi, P\rangle_{\mathsf{var}} \to \langle \phi \mid \mathbb{E}, 0, \star, P\rangle_{\mathsf{var}} \tag{O18}$$
$$\langle \epsilon_1 \circ \epsilon_2 \mid \mathbb{E}, n, \mathcal{E}, P\rangle_{\mathsf{var}} \to \langle \epsilon_1 \mid \mathbb{E}, n, \epsilon_2 \bullet \mathcal{E}, P\rangle_{\mathsf{var}} \tag{O19}$$
$$\langle \phi \mid \mathbb{E}, m, \mathcal{E}, (n, \epsilon)\rangle_{\mathsf{var}} \to \langle \mathbb{E} \mid n[\epsilon]^{\mathsf{ev}}\rangle_{\mathsf{bev}} \quad \text{otherwise} \tag{O20}$$

$$\langle \square\ l :: \mathbb{E} \mid c\rangle_{\mathsf{rec}} \to \langle \mathbb{E} \mid c \mathbin{@^\emptyset} l\rangle_{\mathsf{rec}} \tag{O21}$$
$$\langle l\ \square :: \mathbb{E} \mid c\rangle_{\mathsf{rec}} \to \langle \mathbb{E} \mid l \mathbin{@^\emptyset} c\rangle_{\mathsf{rec}} \tag{O22}$$
$$\langle \lambda :: \mathbb{E} \mid c\rangle_{\mathsf{rec}} \to \langle \mathbb{E} \mid \lambda^\emptyset.c\rangle_{\mathsf{rec}} \tag{O23}$$
$$\langle \mathbb{E} \mid c\rangle_{\mathsf{rec}} \to \langle c \mid \mathbb{E}, \star\rangle_{\mathsf{ev}} \tag{O24}$$

Figure 5 Optimized Abstract Machine.

if $\mathcal{E} = \star$, and to $\bot$ otherwise (cf steps D4, D18, and D20). We then define the invert of $\bullet$, written $\bullet^{-1}$, such that $\bullet^{-1}(\top, !\epsilon_1 \circ \epsilon_2) = (\epsilon_1 \circ \epsilon_2, \star)$ and $\bullet^{-1}(\bot, !\epsilon_1 \circ \epsilon_2) = (\epsilon_1, !\epsilon_2)$, and use it in the backtracking steps D10, D26, and D28. Defining $\bullet^{-1}$ or $\overset{?}{=}$ cannot be done for any non injective operator, but if they exist, such operations could be provided to the automatic derivation procedure with the zipper semantics.

3.2 Optimized Abstract Machine

Being the result of a generic procedure, the DAM is not optimized for the particular case of the λ-calculus with environments. We explain how to optimize it, resulting in the Optimized Abstract Machine, defined in Figure 5.

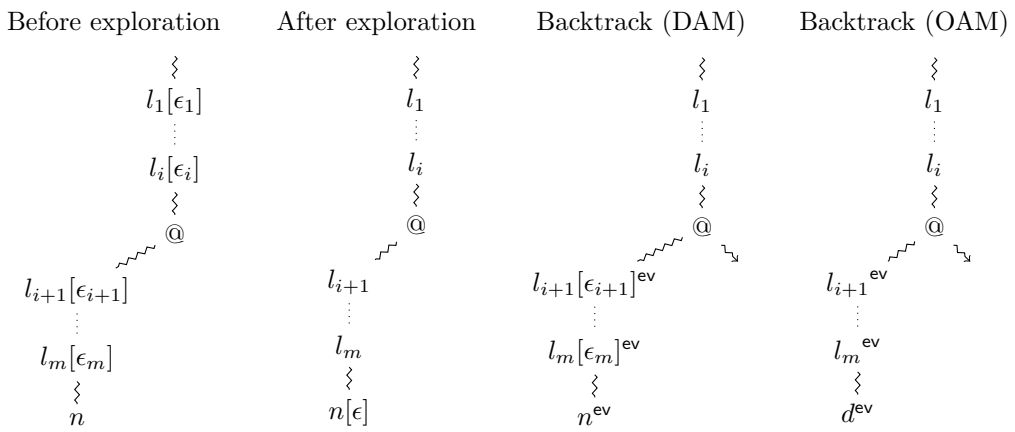


Figure 6 Differences between the DAM and OAM when backtracking.

Bypassing the bvar mode. The DAM does not exploit the fact that solving n in ϵ is syntax-directed: at most one zipper rule applies to a given n and ϵ. Suppose the machine configuration $\mathcal{D} = \langle n^{\Sigma} \mid \mathbb{E}, !\epsilon, \pi \rangle_{\mathsf{ev}}$ initiates a sequence of steps $\mathcal{D} \to^m \mathcal{D}'_{\mathsf{var}}$ such that no forward step applies to $\mathcal{D}'_{\mathsf{var}}$. The DAM then backtracks: starting with $\mathcal{D}'_{\mathsf{var}}$ and for each step, it applies the switching step to add the annotation, and then a backward step. In the end, we need $2m$ steps to add the annotations and get back to $\mathcal{D}$. Instead, the switching step from $\mathcal{D}^m_{\mathsf{var}}$ could restore $\mathcal{D}$ directly, meaning that we no longer need backtracking steps for the var mode, nor annotations for environments.

To implement this change, whenever we start solving n inside ϵ in the OAM, we remember the initial values of the variable and the environment as a pair (n, ϵ) in the var mode (step O5). We let P range over such pairs. The pair is passed along in steps O13 to O19, and is discarded if the variable can in fact be solved in step O14. If variable resolution fails, the "otherwise" step restores the initial values as a closure $n[\epsilon]^{\mathsf{ev}}$ annotated as a normal form (step O20).

Refocusing. The DAM is not in refocused form: after β-reduction, it plugs the resulting closure into $\mathbb{E}$ before looking for the next redex. This is necessary for the machine to be complete w.r.t. the zipper semantics, so that it is able to reach any available redex, including those in $\mathbb{E}$. A middle ground between performance and completeness is to reconstruct an arbitrary part of the context $\mathbb{E}$, while still having the possibility to recreate it completely.

In the OAM, β-reduction produces a closure in the context $\mathbb{E}$ (step O6). The OAM then enters the rec mode dedicated to partially reconstruct the full closure using $\mathbb{E}$. The steps O21 to O23 push the current top frame of the context on the closure. At any point, step O24 may conclude the reconstruction and restart the machine in the ev mode to find the next redex. The reconstruction being non-deterministic itself, the machine may reconstruct fully using the whole context, meaning that any redex in $\mathbb{E}$ is still reachable, and the machine is still complete w.r.t. the zipper semantics.

Removing the stack π. The stack π is used for backtracking, mainly to remember in which order to undo forward steps. To simplify the machine, we would like to use the context $\mathbb{E}$ for that role, as it uniquely changes when the rules appL, appR, and lam are applied. However, it stays the same when env or var are applied (cf. Figure 2). The idea is then to undo the former steps, but not the latter ones. Such a change entails significant differences in the behavior of the OAM compared to the DAM, that we illustrate on an example in Figure 6.

The first picture represents a possible exploration path in a closure, which goes through several constructs $l_1[\epsilon_1] \dots l_m[\epsilon_m]$. Along this path, the two machines collect $\epsilon_1 \dots \epsilon_m$ and combine them into an environment ϵ (second picture).

If solving n in ϵ produces a normal form d, the DAM restores the closure in its original shape while adding annotations, up to some application where it stops backtracking and starts exploring another path. Concretely, the DAM undoes the variable resolution steps, annotates n as a normal form, and puts back the environments $(\epsilon_i \dots \epsilon_m)$ at their original position. The OAM backtracks up to the same application, annotating all the constructs on the way, but without undoing the variable resolution steps – leaving d in place – or restoring $(\epsilon_i \dots \epsilon_m)$.

In practice, the OAM does not feature the stack π, and there are only three backtracking steps (O9 to O11) which mirror the steps O1 to O3. The steps O4 and O5 cannot be reversed. A construct $l[\epsilon]^\Sigma$ in the OAM is therefore either yet to be traversed, or is of the form $n[\epsilon]^{\mathsf{ev}}$ after a failed variable resolution. If we go through a plain term (without any $l[\epsilon]^\Sigma$), we might reach a variable n with the local environment $\star$. To be uniform, we create an annotated closure $n[\mathtt{id}]^{\mathsf{ev}}$ (step O7), turning the plain term itself into a closure. Unlike in the DAM, plain terms cannot be annotated in the OAM: only closures are annotated.

Keeping annotations. Instead of erasing all the annotations after β-reduction as in the DAM, we would like to keep as much of them as possible. Suppose the β-redex is $(\lambda^\emptyset.l) \mathbin{@^\emptyset} k$ in some context $\mathbb{E}$ and current environment $\mathcal{E}$. The normal forms in $\mathbb{E}$ are still normal form after the β-reduction, so if any of them are annotated as such, we should keep these annotations. In contrast, the annotations in the body of the λ-abstraction l may no longer be valid: a term $0^{\mathsf{ev}} \mathbin{@^{\mathsf{ev}}} t$ is no longer a normal form if the β-reduction replaces 0 with a λ-abstraction. As an approximation, we ignore all the annotations inside l.

We therefore need a way to know if an annotation is in the body of a β-redex. We notice that if solving a variable in a local environment $\mathcal{E}$ produces a normal form, then the machine adds an annotation and resets $\mathcal{E}$ in the configuration to $\star$ (steps O20 or O7). We then propagate the annotation upward from the leaves with the combinations of step O8 and steps O9 to O11. In that process, the local environment remains $\star$. We have $\mathcal{E} \neq \star$ only when we explore a closure for which we do not know yet if it is a normal form. This observation is enough to know if an annotation is meaningful after a β-reduction. Let c such that $\mathsf{an}(c) = \mathsf{ev}$ in some environment $\mathcal{E}$; the annotation is valid only if $\mathcal{E} = \star$. Otherwise, the environment has changed after c has been annotated and c is in the body of a β-redex.

On top of that, we need to consider a particular case. Suppose we look for a redex in $(\lambda^\emptyset.0 \mathbin{@^\emptyset} l) \mathbin{@^\emptyset} \lambda^\emptyset.k$, starting with the λ-abstraction on the right, annotating it as $(\lambda^\emptyset.0 \mathbin{@^\emptyset} l) \mathbin{@^\emptyset} \lambda^{\mathsf{ev}}.k$. After β-reduction and solving 0, we get $(\lambda^{\mathsf{ev}}.k) \mathbin{@^\emptyset} l[\epsilon]^\emptyset$ in the environment $\star$ (with $\epsilon = \lambda^{\mathsf{ev}}.k \cdot \mathtt{id}$); according to the previous criterion, the annotation of the λ-abstraction is valid, preventing the machine to β-reduce further. For a λ-abstraction, to be a normal form means it is not in the function position of an application. If a λ-abstraction ends up in such a position after a β-reduction, we need to ignore its annotation.

Consider a local environment $\mathcal{E}$ and a term or closure l. We use booleans $\top, \bot$, ranged over by b, to indicate whether l is in the function position of an application. Given a context $\mathbb{E}$, the function $\mathsf{infun}(\mathbb{E})$ returns $\top$ if the hole of the context is in function position ($\mathbb{E} = \Box\, l :: \mathbb{E}'$), and $\bot$ otherwise. We write $\mathcal{E}, b \vdash l$ if the annotation of l is still valid, i.e., with $\mathcal{E} = \star$, and l is a closure c such that $\mathsf{an}(c) = \mathsf{ev}$, and in the case where l is a λ-abstraction, with $b = \bot$. Otherwise, we write $\mathcal{E}, b \nvdash l$. We use this predicate as a side condition of the steps O1 to O3 and step O14, meaning that we can take these steps if the new subterm under focus is not annotated, or its annotation is meaningless. There is no side condition for step O4, because for this step, the local environment $\epsilon \bullet \mathcal{E}$ cannot be $\star$ by definition of $\bullet$.

Example. We show a machine run starting from $t = (\lambda^{\emptyset}.0\,@^{\emptyset}\,\lambda^{\emptyset}.1)\,@^{\emptyset}(\lambda^{\emptyset}.0)\,@^{\emptyset}\,1$, i.e., $(\lambda x.x\ \lambda y.x)\ (\lambda x.x)\ z$. To simplify, we omit empty annotations and the symbol @ if it is not annotated, writing for instance t as $(\lambda.0\ \lambda.1)\ (\lambda.0)\ 1$

The machine may go right of the topmost application, focusing on an unsolvable variable, since $\mathcal{E}$ is $\star$ for now. The machine annotates it and backtracks.

$$\langle t \mid \blacksquare, \star\rangle_{\mathsf{ev}} \overset{\mathrm{O2}}{\to} \langle 1 \mid (\lambda.0\ \lambda.1)\ (\lambda.0)\ \square :: \blacksquare, \star\rangle_{\mathsf{ev}} \overset{\mathrm{O7}}{\to} \langle (\lambda.0\ \lambda.1)\ (\lambda.0)\ \square :: \blacksquare \mid 1[\mathtt{id}]^{\mathsf{ev}}\rangle_{\mathsf{bev}}$$
$$\overset{\mathrm{O10}}{\to} \langle (\lambda.0\ \lambda.1)\ (\lambda.0)\ 1[\mathtt{id}]^{\mathsf{ev}} \mid \blacksquare, \star\rangle_{\mathsf{ev}} \triangleq \mathcal{O}_1$$

The machine has no choice but to go left of the application. Then it might go left again, and inside the body of the λ-abstraction.

$$\mathcal{O}_1 \overset{\mathrm{O1}}{\to}\overset{\mathrm{O1}}{\to}\overset{\mathrm{O3}}{\to} \langle 0\ \lambda.1 \mid \lambda :: \square\ \lambda.0 :: \square\ 1[\mathtt{id}]^{\mathsf{ev}} :: \blacksquare, \star\rangle_{\mathsf{ev}} \triangleq \mathcal{O}_2$$

The term $0\ \lambda.1$ is a normal form, so the machine annotates it and backtracks.

$$\mathcal{O}_2 \to^* \overset{\mathrm{O8}}{\to} \langle \lambda :: \square\ \lambda.0 :: \square\ 1[\mathtt{id}]^{\mathsf{ev}} :: \blacksquare \mid 0[\mathtt{id}]^{\mathsf{ev}}\ @^{\mathsf{ev}}\ \lambda^{\mathsf{ev}}.1[\mathtt{id}]^{\mathsf{ev}}\rangle_{\mathsf{bev}}$$
$$\overset{\mathrm{O11}}{\to} \langle \lambda.0[\mathtt{id}]^{\mathsf{ev}}\ @^{\mathsf{ev}}\ \lambda^{\mathsf{ev}}.1[\mathtt{id}]^{\mathsf{ev}} \mid \square\ \lambda.0 :: \square\ 1[\mathtt{id}]^{\mathsf{ev}} :: \blacksquare, \star\rangle_{\mathsf{ev}} \triangleq \mathcal{O}_3$$

At this point, the machine β-reduces and switches to the rec mode, in which it may decide to reconstruct the frame in the context. It then restarts in ev mode to find the next redex.

$$\mathcal{O}_3 \overset{\mathrm{O6}}{\to} \langle \square\ 1[\mathtt{id}]^{\mathsf{ev}} :: \blacksquare \mid (0[\mathtt{id}]^{\mathsf{ev}}\ @^{\mathsf{ev}}\ \lambda^{\mathsf{ev}}.1[\mathtt{id}]^{\mathsf{ev}})[\lambda.0 \cdot \mathtt{id}]\rangle_{\mathsf{rec}}$$
$$\overset{\mathrm{O22}}{\to} \langle \blacksquare \mid (0[\mathtt{id}]^{\mathsf{ev}}\ @^{\mathsf{ev}}\ \lambda^{\mathsf{ev}}.1[\mathtt{id}]^{\mathsf{ev}})[\lambda.0 \cdot \mathtt{id}]\ 1[\mathtt{id}]^{\mathsf{ev}}\rangle_{\mathsf{rec}}$$
$$\overset{\mathrm{O24}}{\to} \langle (0[\mathtt{id}]^{\mathsf{ev}}\ @^{\mathsf{ev}}\ \lambda^{\mathsf{ev}}.1[\mathtt{id}]^{\mathsf{ev}})[\lambda.0 \cdot \mathtt{id}]\ 1[\mathtt{id}]^{\mathsf{ev}} \mid \blacksquare, \star\rangle_{\mathsf{ev}} \triangleq \mathcal{O}_4$$

The machine cannot go right of the application, because we have $\star, \bot \vdash 1[\mathtt{id}]^{\mathsf{ev}}$: the annotation is still valid. The machine therefore has to go left.

$$\mathcal{O}_4 \overset{\mathrm{O1}}{\to} \langle (0[\mathtt{id}]^{\mathsf{ev}}\ @^{\mathsf{ev}}\ \lambda^{\mathsf{ev}}.1[\mathtt{id}]^{\mathsf{ev}})[\lambda.0 \cdot \mathtt{id}] \mid \square\ 1[\mathtt{id}]^{\mathsf{ev}} :: \blacksquare, \star\rangle_{\mathsf{ev}}$$
$$\overset{\mathrm{O4}}{\to} \langle 0[\mathtt{id}]^{\mathsf{ev}}\ @^{\mathsf{ev}}\ \lambda^{\mathsf{ev}}.1[\mathtt{id}]^{\mathsf{ev}} \mid \square\ 1[\mathtt{id}]^{\mathsf{ev}} :: \blacksquare, !\lambda.0 \cdot \mathtt{id}\rangle_{\mathsf{ev}} \triangleq \mathcal{O}_5$$

Because the local environment is no longer $\star$, the annotations of the subterm under focus are not valid and can be ignored. For instance, we can go right of the application and under the λ-abstraction to reach the variable.

$$\mathcal{O}_5 \overset{\mathrm{O2}}{\to}\overset{\mathrm{O3}}{\to}\overset{\mathrm{O4}}{\to} \langle 1 \mid \lambda :: 0[\mathtt{id}]^{\mathsf{ev}}[\lambda.0 \cdot \mathtt{id}]\ \square :: \square\ 1[\mathtt{id}]^{\mathsf{ev}} :: \blacksquare, !\mathtt{id} \circ \Uparrow(\lambda.0 \cdot \mathtt{id})\rangle_{\mathsf{ev}} \triangleq \mathcal{O}_6$$

Solving the variable in the var mode produces $\lambda.0$, a normal form, annotated as such by the machine, which then goes back to the first application in the context.

$$\mathcal{O}_6 \to^* \langle \lambda.0 \mid \lambda :: 0[\mathtt{id}]^{\mathsf{ev}}[\lambda.0 \cdot \mathtt{id}]\ \square :: \square\ 1[\mathtt{id}]^{\mathsf{ev}} :: \blacksquare, !\uparrow\rangle_{\mathsf{ev}}$$
$$\to^* \langle \lambda.0[\Uparrow\uparrow]^{\mathsf{ev}} \mid \lambda :: 0[\mathtt{id}]^{\mathsf{ev}}[\lambda.0 \cdot \mathtt{id}]\ \square :: \square\ 1[\mathtt{id}]^{\mathsf{ev}} :: \blacksquare, \star\rangle_{\mathsf{ev}}$$
$$\to^* \langle 0[\mathtt{id}]^{\mathsf{ev}}[\lambda.0 \cdot \mathtt{id}]\ \lambda^{\mathsf{ev}}.\lambda^{\mathsf{ev}}.0[\Uparrow\uparrow]^{\mathsf{ev}} \mid \square\ 1[\mathtt{id}]^{\mathsf{ev}} :: \blacksquare, \star\rangle_{\mathsf{ev}} \triangleq \mathcal{O}_7$$

The machine has no choice but to go left of the application, until it reaches the variable, which can be solved.

$$\mathcal{O}_7 \overset{\mathrm{O1}}{\to}\overset{\mathrm{O4}}{\to}\overset{\mathrm{O4}}{\to} \langle 0 \mid \square\ \lambda^{\mathsf{ev}}.\lambda^{\mathsf{ev}}.0[\Uparrow\uparrow]^{\mathsf{ev}} :: \square\ 1[\mathtt{id}]^{\mathsf{ev}} :: \blacksquare, !\mathtt{id} \circ (\lambda.0 \cdot \mathtt{id})\rangle_{\mathsf{ev}}$$
$$\overset{\mathrm{O5}}{\to}\overset{\mathrm{O19}}{\to}\overset{\mathrm{O15}}{\to}\overset{\mathrm{O14}}{\to} \langle \lambda.0 \mid \square\ \lambda^{\mathsf{ev}}.\lambda^{\mathsf{ev}}.0[\Uparrow\uparrow]^{\mathsf{ev}} :: \square\ 1[\mathtt{id}]^{\mathsf{ev}} :: \blacksquare, \star\rangle_{\mathsf{ev}} \triangleq \mathcal{O}_8$$

The machine may then β-reduce and not reconstruct anything from the context.

$$\mathcal{O}_8 \overset{\mathrm{O6O24}}{\to}\overset{\mathrm{O6O24}}{\to} \langle 0[\lambda^{\mathsf{ev}}.\lambda^{\mathsf{ev}}.0[\Uparrow\uparrow]^{\mathsf{ev}} \cdot \mathtt{id}] \,|\, \Box\, 1[\mathtt{id}]^{\mathsf{ev}} :: \blacksquare, \star\rangle_{\mathsf{ev}} \triangleq \mathcal{O}_9$$

The variable can be solved immediately, producing a λ-abstraction. A β-reduction is then possible, without any reconstruction because the context is empty.

$$\mathcal{O}_9 \to^* \langle \lambda^{\mathsf{ev}}.\lambda^{\mathsf{ev}}.0[\Uparrow\uparrow]^{\mathsf{ev}} \,|\, \Box\, 1[\mathtt{id}]^{\mathsf{ev}} :: \blacksquare, \star\rangle_{\mathsf{ev}} \to^* \langle (\lambda^{\mathsf{ev}}.0[\Uparrow\uparrow]^{\mathsf{ev}})[1[\mathtt{id}]^{\mathsf{ev}} \cdot \mathtt{id}] \,|\, \blacksquare, \star\rangle_{\mathsf{ev}}$$

The machine has to go through the resulting closure to annotate it as a normal form, pushing the environment $1[\mathtt{id}]^{\mathsf{ev}} \cdot \mathtt{id}$ in the body of the λ-abstraction while doing so. The run ends with $\langle \lambda^{\mathsf{ev}}.0[\Uparrow(1[\mathtt{id}]^{\mathsf{ev}} \cdot \mathtt{id}) \circ \Uparrow\uparrow]^{\mathsf{ev}} \rangle_{\mathsf{nf}}$.

Implementation. We have implemented the OAM as a skeletal semantics [20] from which we extract an OCaml program that evaluates all possible reductions from a starting term. The implementation is very close to the formal definition of the OAM and is available online [13].

3.3 Correspondence between the DAM and the OAM

To relate the OAM and the DAM, we define an equivalence $\approx$ between OAM and DAM machine states, and prove that from a sequence of OAM steps, we can build a sequence of equivalent DAM steps and conversely. We let $\mathcal{O}$ and $\mathcal{D}$ range over the machines states of the optimized and the derived machines, respectively. We informally explain the ingredients needed to define $\approx$; the formal definitions and proofs are in the report [15, Appendix B].

Consider two similar runs of the OAM and DAM from the same starting point: after doing the same β-reductions, they are at the same position in the closure. As explained in Figure 6 some environments may be pushed to the leaves and some variables may be solved in the OAM, but not in the DAM. Besides the closures on the OAM side may contain more annotations than on the DAM one, but some of these annotations could be meaningless.

We relate two states $\mathcal{O}$ and $\mathcal{D}$ by pushing all the environments to the leaves, and then solving all the variables, which is exactly what the translation $[\![\cdot]\!]$ of Figure 3 is doing. We define two similar translations on annotated closures or terms $[\![\cdot]\!]_{\mathsf{o}}$ (for the OAM) and $[\![\cdot]\!]_{\mathsf{d}}$ (for the DAM) which produce annotated terms. The translation $[\![\cdot]\!]_{\mathsf{o}}$ also removes meaningless annotations, by using the same criterion as in Section 3.2. We then compare the results r_{o} and r_{d} of these translations using a predicate $\triangleright$, which checks that the two annotated terms are the same, except that r_{o} may contain more annotations than r_{d}.

We extend $[\![\cdot]\!]_{\mathsf{o}}$, $[\![\cdot]\!]_{\mathsf{d}}$ and $\triangleright$ to contexts frame by frame, and we write $\mathcal{O} \approx \mathcal{D}$ if either:

- $\mathcal{O} = \langle l \,|\, \mathbb{E}, \mathcal{E}\rangle_{\mathsf{ev}}$, $\mathcal{D} = \langle k \,|\, \mathbb{F}, \mathcal{F}, \pi\rangle_{\mathsf{ev}}$, $[\![l]\!]_{\mathsf{o}}(\mathcal{E}) \triangleright [\![k]\!]_{\mathsf{d}}(\mathcal{F})$, and $[\![\mathbb{E}]\!]_{\mathsf{o}} \triangleright [\![\mathbb{F}]\!]_{\mathsf{d}}$;
- $\mathcal{O} = \langle \mathbb{E} \,|\, c\rangle_{\mathsf{bev}}$, $\mathcal{D} = \langle \pi \,|\, k, \mathbb{F}, \mathcal{F}\rangle_{\mathsf{bev}}$, $[\![c]\!]_{\mathsf{o}}(\star) \triangleright [\![k]\!]_{\mathsf{d}}(\mathcal{F})$, and $[\![\mathbb{E}]\!]_{\mathsf{o}} \triangleright [\![\mathbb{F}]\!]_{\mathsf{d}}$;
- $\mathcal{O} = \langle c\rangle_{\mathsf{nf}}$, $\mathcal{D} = \langle d\rangle_{\mathsf{nf}}$, and $[\![c]\!]_{\mathsf{o}}(\star) \triangleright [\![d]\!]_{\mathsf{d}}(\star)$.

The equivalence is defined only on the modes which are common to both machines. Equivalent machine states can be considered as synchronization points between the OAM and DAM machine runs. In particular, the contexts $\mathbb{E}$ and $\mathbb{F}$ of two equivalent states have the same shape, so the machines are at the same position in the term or closure being explored.

We extend the equivalence to sequences of machines states, writing $(\mathcal{O}_0, \mathcal{O}_1 \dots \mathcal{O}_i) \approx (\mathcal{D}_0, \mathcal{D}_1 \dots \mathcal{D}_i)$ if $\mathcal{O}_0 = \langle l, \blacksquare, \star\rangle_{\mathsf{ev}}$, $\mathcal{D}_0 = \langle l, \blacksquare, \star, \blacktriangle\rangle_{\mathsf{ev}}$ for the same l, $\mathcal{O}_j \to^* \mathcal{O}_{j+1}$ and $\mathcal{D}_j \to^* \mathcal{D}_{j+1}$ for all $0 \leq j < i$, and $\mathcal{O}_j \approx \mathcal{D}_j$ for all $0 \leq j \leq i$. The correspondence theorems then state that given two equivalent sequences, if one of the machines is doing a step, then we can find equivalent states to continue the sequences.

▶ **Theorem 3** (OAM implies DAM). *If* $(\mathcal{O}_0, \mathcal{O}_1 \dots \mathcal{O}_i) \approx (\mathcal{D}_0, \mathcal{D}_1 \dots \mathcal{D}_i)$ *and* $\mathcal{O}_i \to \mathcal{O}'$*, there exist* $\mathcal{O}_{i+1}$ *and* $\mathcal{D}_{i+1}$ *such that* $\mathcal{O}' \to^* \mathcal{O}_{i+1}$*,* $\mathcal{D}_i \to^* \mathcal{D}_{i+1}$*, and* $\mathcal{O}_{i+1} \approx \mathcal{D}_{i+1}$*.*

▶ **Theorem 4** (DAM implies OAM). *If* $(\mathcal{O}_0, \mathcal{O}_1 \dots \mathcal{O}_i) \approx (\mathcal{D}_0, \mathcal{D}_1 \dots \mathcal{D}_i)$ *and* $\mathcal{D}_i \to \mathcal{D}'$*, there exist* $\mathcal{O}_{i+1}$ *and* $\mathcal{D}_{i+1}$ *such that* $\mathcal{D}' \to^* \mathcal{D}_{i+1}$*,* $\mathcal{O}_i \to^* \mathcal{O}_{i+1}$*, and* $\mathcal{O}_{i+1} \approx \mathcal{D}_{i+1}$*.*

We sketch some of the cases of Theorem 3, the proofs for Theorem 4 are similar. Step O20 of the OAM (Figure 5) is matched in the DAM by a sequence of bvar steps which produces an equivalent configuration. Applying the axiom (step O6) leads to the rec mode, in which a number m of frames of $\mathbb{E}$ are reconstructed. If $\mathbb{E}$ contains n frames in total, the DAM matches by applying its β-reduction step (step D6 in Figure 4), which rebuilds the whole context, and then it gets to the same position as in the OAM with $n - m$ forward steps.

A backward step of the OAM, e.g. step O9, is not necessarily matched by the corresponding step in the DAM – step D8 in Figure 4 – because the DAM may have to undo steps corresponding to the applications of the rules env or var. We prove that we can undo them first and then apply step D8 in the DAM. Finally, suppose a valid annotation in the OAM has been erased at the same position in the DAM. The subterm under focus is still a normal form, so the sequence which added the annotation on the DAM side before its erasure can still be applied to that subterm to restore the annotation.

4 Recovering Known Strategies and Abstract Machines

The OAM can be used to obtain specialized abstract machines that implement various reduction strategies. A reduction strategy can be seen as a way to choose a particular redex. In the OAM, the transitions O1, O2, O3 are navigating through the term or closure by going left ($\swarrow$) or right ($\searrow$) of the application, or down under the lambda ($\downarrow$), respectively. By controlling which of these navigation rules to apply and in which order, we can define various deterministic or non-deterministic strategies. To define reduction strategies succinctly, we use the phased format of strategies [18]. For example, the call-by-name strategy can be defined as $cbn = \swarrow cbn; \beta$, a strategy that navigates to the left of an application as much as possible (iterating itself), and then tries to perform β-reduction. Formally, a strategy can be defined as a set of term decompositions. The sequence of strategies $r; s$ is a strategy containing any decomposition in r, or a r-normal form that is in s. Furthermore, $r \cup s$ is a strategy that contains any decomposition that is in r or in s. If a term admits multiple decompositions, according to r or to s, the strategy $r \cup s$ is non-deterministic.

Krivine Abstract Machine. The Krivine Abstract Machine [38] implements exactly call-by-name evaluation as defined above. In the OAM, it corresponds to restricting navigation to only $\swarrow$, and removing $\searrow$ and $\downarrow$. Thus, we go left as much as possible, and if we reach a λ-abstraction, we try to apply step O6 to β-reduce. If it fails because the context is empty, we are done, and we apply steps O8 and O12: the (annotated) lambda abstraction will be returned as the normal form. If, on the other hand, we reach a variable, the machine tries to solve it by rule O5. If it is a free variable, the machine cannot solve it and backtracks, thus reconstructing the call-by-name normal form – an application of a (annotated) variable to a number of closures (possibly none). An inspection of reachable machine configurations reveals that excluding the two navigation rules further entails that rules O10, O11, and also O16-O18, O22, O23 are not reachable: environments are constructed using only two operators, and thus the machine is significantly simplified. Moreover, the call-by-name strategy being deterministic, the reconstruction transitions can be short-circuited and eliminated altogether.

Right-to-left open call by value. We consider a variant of the call-by-value strategy that operates on open terms, and evaluates arguments in an application from right to left: it is defined as $rcbw = \searrow rcbw; \swarrow rcbw; \beta$. This strategy can be realized in the OAM by keeping $\swarrow$ and $\searrow$ and forbidding $\downarrow$, but with the priority given to $\searrow$, applying it whenever possible before we try $\swarrow$. Whenever we can apply the β-rule, the argument on top of the stack must be a value, because of the precedence of $\searrow$ over $\swarrow$: the machine only performs a restricted β-contraction, as in call-by-value. The resulting machine is closest to the variant presented in [19], or it can be seen as a right-to-left variant of the CEK machine [32].

In contrast, we cannot obtain the left-to-right variant of call-by-value in a similar way. To do it, we would need to prioritize $\swarrow$ over $\searrow$, which in turn requires further inspection of the top of the stack to make sure that β-contraction is only applied when the argument is a value. Other similar call-by-value strategies [43, 7] cannot be expressed. It is due to the fact that OAM only looks at the top-level constructor and no further.

Normal order. The normal-order strategy is a fully reducing strategy that can be seen as an iteration of call-by-name until no redex is left anywhere in the term. Its phased form is $no = (\beta; \swarrow no; \searrow no) \cup \downarrow no$. It is worth noting that in this case the sum of strategies $\cup$ does not produce a non-deterministic strategy, because $(\beta; \swarrow no; \searrow no)$ and $\downarrow no$ each contains decompositions of structurally different terms. Normal order is an example of a hybrid strategy that does not behave uniformly, but the navigation steps depend on the context: the $\downarrow$ rule can only be applied when the lambda abstraction is not in the operator position. A canonical abstract machine that implements normal order is Crégut's KN machine [23]. In the OAM, we first try to apply β, then $\swarrow$, then $\searrow$. Otherwise, we can try $\downarrow$ (only when we know that β is not applicable). This adjustment of OAM results in a variant of KN.

Weak reduction. A simple restriction that results in a non-deterministic strategy can be obtained by removing the rule $\downarrow$ from the OAM. This strategy can be described in the phased form as $weak = \swarrow weak \cup \searrow weak \cup \beta$.

Head reduction. Consider the following two strategies in phased form: $head = (\beta; \swarrow head) \cup \downarrow head$ and $ihead = (\swarrow ihead; \beta) \cup \downarrow ihead$. They both perform head reduction and compute head normal forms (i.e., terms of the form $\lambda x_1 \ldots \lambda x_n . y\ t_1 \ldots t_n$), but the first strategy is "outermost", and the second is "innermost". The two strategies are deterministic, because for each term only one of the $\cup$-substrategies can be used. If we restrict OAM in that we remove the rule $\searrow$ and order the remaining evaluation rules: first $\swarrow$, $\downarrow$, and then β, we obtain the implementation of the *ihead* strategy. If, on the other hand, we put these rules in the order: β first, then $\swarrow$, then $\downarrow$, we obtain the machine for the *head* strategy. In both cases, we can further trim the machine by removing all the transitions made unreachable by removing $\searrow$.

Unexpressible strategies. We have remarked above that the left-to-right call by value is not expressible in an instance of OAM obtained by restriction and prioritization only. The same is true for any strategy that hinges on the same issue, like the *fireball calculus* [43, 7], the non-deterministic variant of CbV where we can reduce freely to the left or to the right of the application as long as it is possible, and only then we try to β-reduce. This strategy in phased form is defined as $cbw = (\swarrow cbw \cup \searrow cbw); \beta$, but in fact this definition enforces that β-contraction will be applied only when the argument is a weak normal form. So it is not possible to directly obtain this kind of behavior in the OAM without further inspection of the stack in O6, as in the left-to-right call by value.

5 Related Work

Non-deterministic abstract machines for the λ-calculus. In [14], Biernacka et al. present a general framework for deriving non-deterministic abstract machines from zipper semantics. One of the example derivations in their work leads to a non-deterministic abstract machine for the λ-calculus, where no reduction strategy is imposed and each redex in a given term can be selected for contraction thanks to a loop-free backtracking mechanism encoded in the transition system of the machine. The abstract machines of the present work follow the same principles, but they are optimized crucially in two ways: they replace substitution with environments and they are in the refocused form. Both these advancements greatly improve the performance measures of the machines since they influence, respectively, how substitution is implemented [6] and how term decomposition is resumed after a redex has been contracted [27, 28].

A lightweight form of non-determinism is present in [3], a recent work by Accattoli and Barenbaum who introduce an abstract machine for normal-order reduction in the λ-calculus that has a diamond property. Their machine is non-deterministic in that when dealing with a partially normalized term of the form $x\,t_1\,\ldots\,t_n$, it can choose arbitrarily which argument t_i to reduce next. (The machine can also jump from one job to another when a normal form has been reached, without a step-by-step backtracking.) The machine is diamond non-deterministic, i.e., it satisfies the strongest, one-step form of confluence, which implies that the choice of a reduction site does not affect the result, nor the length of the machine run. The machines considered in the present work, as more general, obviously are not diamond non-deterministic in this sense.

Explicit substitutions and environment-based abstract machines. In their pioneering work on explicit substitutions, Abadi et al. [1] present an abstract machine for weak-head normalization in the $\lambda\sigma$-calculus that generalizes the KAM (originally featuring de Bruijn indices and closures) to open terms. Full normalization is then achieved as an iteration of the weakly normalizing machine. Substitutions from the calculus play the role of environments in the machine, which makes the relationship between the calculus and the machine particularly explicit. The $\lambda\sigma$-calculus is also used by Crégut [23] for designing and establishing the correctness of his strongly normalizing abstract machines KN and KNL. Call-by-value and call-by-name reduction strategies and their implementation on an environment-based abstract machine are discussed in the context of a variant of $\lambda\sigma$ with names and garbage collection by Rose [45].

In [24], Curien introduces a minimal weak non-deterministic calculus of closures $\lambda\rho$, requiring a big-step reduction rule for term application, that allows him to express the reduction strategies for the KAM and the Categorical Abstract Machine (CAM) [22], taking advantage of the presence of closures in the calculus. The author also sketches a generalization of the weak calculus to a strong version that accounts for full β-reduction, and is formalized in [25] as $\lambda\sigma_{\Uparrow}$.

The full power of the $\lambda\sigma_{\Uparrow}$-calculus is then used in Hardin et al.'s framework for reasoning about the correspondence between several weak reduction strategies in $\lambda\sigma_w$, a subcalculus of $\lambda\sigma_{\Uparrow}$, and known abstract machines such as the KAM, SECD, CAM, and Cardelli's FAM [21]. Curien's weak calculus of closures is, in turn, studied by Biernacka and Danvy [19] who introduce its modification $\lambda\hat{\rho}$ that, unlike $\lambda\rho$, makes it possible to define one-step reduction strategies underlying abstract machines such as the KAM, CEK, and Leroy's ZINC [41]. It is shown that $\lambda\hat{\rho}$ and the abstract machines are in an intimate relation established by an enriched refocusing. The calculus $\lambda\hat{\rho}$ is then used by García-Pérez et al. [35, 34] to reconstruct

and prove Cregut's KN correct in a very precise sense. In our case, the underlying small-step semantics is zipper zemantics, rather than the more traditional structural operational semantics.

Accattoli et al. [4] study the relationship between the linear substitution calculus (LSC) [11] and a number of abstract machines for weak normalization, including the KAM, CEK, a simplified version of SECD, the lazy KAM [23], and Sestoft's call-by-need machine [48] (ZINC is also sketched). It is argued that a weak version of LSC, a calculus with an explicit representation of reduction contexts and a notion of (explicit) substitution at a distance, can abstractly and effectively represent or distill, in the authors' words, abstract machines for weak normalization. Interestingly, the authors also introduce new machines inspired by LSC: the Milner Abstract Machine (MAM) that uses only one, global environment and therefore avoids the concept of closure at a cost of explicit α-renaming, and the Milner Abstract machine by-neeD (MAD) that takes advantage of the single global environment introduced in the MAM that facilitates sharing of computed values.

In [5], the above ideas are used to design the Strong Milner Abstract Machine, a machine for normal-order strong normalization that features a single environment and is shown to have a linear overhead both in the number of steps in a reduction sequence in LSC and in the size of the initial term – Strong MAM is distilled (decoded) into LSC and analysed for complexity. Such issues are central in Accattoli's Useful MAM [2], that is the first reasonable, i.e., polynomially related to the cost model of RAM or Turing machines, machine for strong normalization in the λ-calculus in the literature. Useful MAM, however, is not studied in the context of LSC or another calculus of explicit substitutions. In the same line of work, Accattoli and Barras provide a thorough complexity analysis and comparison between machines with local and global environments [6], whereas Accattoli et al. [7, 10], present reasonable abstract machines for open call-by-value that rely on a global environment.

A special form of environments, called crumbled environments, is introduced in [9], where Accattoli et al. study call-by-value evaluation of both closed and open λ-terms. Such environments stem from the representation of terms considered by the authors, i.e., a representation with explicit sharing or explicit substitutions (in a new formulation), and they encode evaluation contexts besides storing delayed substitutions. The crumbling technique is then used by Accattoli et al. to define a reasonable abstract machine, called SCAM, for strong call-by-value [8]. Their work takes advantage of yet another calculus of explicit substitutions – the value substitution calculus (VSC) and of the concept of implosive sharing known from the literature on call-by-need [49]. Other reasonable abstract machines for strong call-by-value and strong call-by-need have been derived through the functional correspondence [12] from higher-order evaluators with heap-based sharing and local environments by Biernacka et al. [16, 17].

In contrast to the above collection of results by Accattoli et al. and Biernacka et al., in the present work we do not consider any sharing-oriented optimizations that go beyond the introduction of local environments, and we focus solely on the interaction of such environments with non-determinism. Whether such optimizations can be incorporated in NDAMs is a worthwhile question that requires further research.

6 Conclusion

We present an environment-based abstract machine for the λ-calculus without a fixed reduction strategy. To this end, we define a zipper semantics – inspired by a calculus of explicit substitutions [25] – and derive an NDAM from it. We then optimize the NDAM so that it requires fewer steps to find a redex. In the resulting OAM, we can recognize existing deterministic machines by restricting and/or prioritizing steps.

Our work demonstrates that zipper semantics can be defined with environments instead of substitution. It also shows that the derived machine can be used as a stepping stone towards more optimized machines. Some optimizations are language specific, like knowing which annotations to keep after reduction, but others could be made generic in the NDAM derivation procedure. For example, it is always possible to bypass the backtracking steps of a deterministic mode of the machine, like we do with the bvar mode in the OAM. Refocusing is also possible as long as the stack π can be replaced by another element for backtracking, such as the context $\mathbb{E}$. In fact, the non-deterministic machines for the λ-calculus with substitution and for HOcore [14] could be automatically derived in the refocused form from their respective zipper semantics.

We believe that our approach of defining a zipper semantics, optimizing the derived machine, and possibly restricting it to make it deterministic is generic and can be used to define non-deterministic and deterministic abstract machines for other calculi. In particular, we would like to apply this technique to the λ-calculus with futures [33, 42], and see whether we can recover the different task creation strategies of this calculus, such as the eager strategy [37] or the lazy one [30].

References

1 Martín Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy. Explicit substitutions. *J. Funct. Program.*, 1(4):375–416, 1991.

2 Beniamino Accattoli. The Useful MAM, a reasonable implementation of the strong λ-calculus. In Jouko A. Väänänen, Åsa Hirvonen, and Ruy J. G. B. de Queiroz, editors, *Logic, Language, Information, and Computation - 23rd International Workshop, WoLLIC 2016, Puebla, Mexico, August 16-19th, 2016. Proceedings*, volume 9803 of *Lecture Notes in Computer Science*, pages 1–21. Springer, 2016. doi:10.1007/978-3-662-52921-8_1.

3 Beniamino Accattoli and Pablo Barenbaum. A diamond machine for strong evaluation. *CoRR*, abs/2309.12515, 2023. doi:10.48550/arXiv.2309.12515.

4 Beniamino Accattoli, Pablo Barenbaum, and Damiano Mazza. Distilling abstract machines. In Johan Jeuring and Manuel M. T. Chakravarty, editors, *Proceedings of the 19th ACM SIGPLAN International Conference on Functional Programming, Gothenburg, Sweden, September 1-3, 2014*, pages 363–376. ACM, 2014. doi:10.1145/2628136.2628154.

5 Beniamino Accattoli, Pablo Barenbaum, and Damiano Mazza. A strong distillery. In Xinyu Feng and Sungwoo Park, editors, *Programming Languages and Systems - 13th Asian Symposium, APLAS 2015, Pohang, South Korea, November 30 - December 2, 2015, Proceedings*, volume 9458 of *Lecture Notes in Computer Science*, pages 231–250. Springer, 2015. doi:10.1007/978-3-319-26529-2_13.

6 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, Namur, Belgium, October 09 - 11, 2017*, pages 4–16. ACM, 2017. doi:10.1145/3131851.3131855.

7 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.

8 Beniamino Accattoli, Andrea Condoluci, and Claudio Sacerdoti Coen. Strong call-by-value is reasonable, implosively. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–14. IEEE, 2021. doi:10.1109/LICS52264.2021.9470630.

9 Beniamino Accattoli, Andrea Condoluci, Giulio Guerrieri, and Claudio Sacerdoti Coen. Crumbling abstract machines. In Ekaterina Komendantskaya, editor, *Proceedings of the 21st International Symposium on Principles and Practice of Programming Languages, PPDP 2019, Porto, Portugal, October 7-9, 2019*, pages 4:1–4:15. ACM, 2019. doi:10.1145/3354166.3354169.

10 Beniamino Accattoli and Giulio Guerrieri. Abstract machines for open call-by-value. *Sci. Comput. Program.*, 184, 2019. doi:10.1016/j.scico.2019.03.002.

11 Beniamino Accattoli and Delia Kesner. The structural *lambda*-calculus. In Anuj Dawar and Helmut Veith, editors, *Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23-27, 2010. Proceedings*, volume 6247 of *Lecture Notes in Computer Science*, pages 381–395. Springer, 2010. doi:10.1007/978-3-642-15205-4_30.

12 Mads Sig Ager, Dariusz Biernacki, Olivier Danvy, and Jan Midtgaard. A functional correspondence between evaluators and abstract machines. In *Proceedings of the 5th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, 27-29 August 2003, Uppsala, Sweden*, pages 8–19. ACM, 2003.

13 Małgorzata Biernacka, Dariusz Biernacki, Sergueï Lenglet, and Alan Schmitt. Optimized abstract machine implementation. Software, swhId: swh:1:dir:ba776d5226cde3b68cc6346fcd3a3495ad4d04e8 (visited on 06/06/2024). URL: https://gitlab.inria.fr/skeletons/lambda-am.git.

14 Małgorzata Biernacka, Dariusz Biernacki, Sergueï Lenglet, and Alan Schmitt. Non-deterministic abstract machines. In Bartek Klin, Slawomir Lasota, and Anca Muscholl, editors, *33rd International Conference on Concurrency Theory, CONCUR 2022, September 12-16, 2022, Warsaw, Poland*, volume 243 of *LIPIcs*, pages 7:1–7:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

15 Małgorzata Biernacka, Dariusz Biernacki, Sergueï Lenglet, and Alan Schmitt. Optimizing a non-deterministic abstract machine with environments. Available at https://inria.hal.science/hal-04568253, 2024.

16 Małgorzata Biernacka, Witold Charatonik, and Tomasz Drab. A derived reasonable abstract machine for strong call by value. In Niccolò Veltri, Nick Benton, and Silvia Ghilezan, editors, *PPDP 2021: 23rd International Symposium on Principles and Practice of Declarative Programming, Tallinn, Estonia, September 6-8, 2021*, pages 6:1–6:14. ACM, 2021. doi:10.1145/3479394.3479401.

17 Małgorzata Biernacka, Witold Charatonik, and Tomasz Drab. A simple and efficient implementation of strong call by need by an abstract machine. *Proc. ACM Program. Lang.*, 6(ICFP):109–136, 2022. doi:10.1145/3549822.

18 Małgorzata Biernacka, Witold Charatonik, and Tomasz Drab. The zoo of lambda-calculus reduction strategies, and Coq. In June Andronick and Leonardo de Moura, editors, *13th International Conference on Interactive Theorem Proving, ITP 2022, August 7-10, 2022, Haifa, Israel*, volume 237 of *LIPIcs*, pages 7:1–7:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ITP.2022.7.

19 Małgorzata Biernacka and Olivier Danvy. A concrete framework for environment machines. *ACM Trans. Comput. Log.*, 9(1):6, 2007.

20 Martin Bodin, Philippa Gardner, Thomas Jensen, and Alan Schmitt. Skeletal semantics and their interpretations. *PACMPL*, 3(POPL):44:1–44:31, 2019.

21 Luca Cardelli. Compiling a functional language. In Robert S. Boyer, Edward S. Schneider, and Guy L. Steele Jr., editors, *Proceedings of the 1984 ACM Conference on LISP and Functional Programming, LFP 1984, Austin, Texas, USA, August 5-8, 1984*, pages 208–217. ACM, 1984. doi:10.1145/800055.802037.

22 Guy Cousineau, Pierre-Louis Curien, and Michel Mauny. The Categorical Abstract Machine. *Sci. Comput. Program.*, 8(2):173–202, 1987. doi:10.1016/0167-6423(87)90020-7.

23 Pierre Crégut. Strongly reducing variants of the Krivine abstract machine. *High. Order Symb. Comput.*, 20(3):209–230, 2007. doi:10.1007/s10990-007-9015-z.

24 Pierre-Louis Curien. An abstract framework for environment machines. *Theor. Comput. Sci.*, 82(2):389–402, 1991. doi:10.1016/0304-3975(91)90230-Y.

25 Pierre-Louis Curien, Thérèse Hardin, and Jean-Jacques Lévy. Confluence properties of weak and strong calculi of explicit substitutions. *J. ACM*, 43(2):362–397, 1996. doi:10.1145/226643.226675.

26 Olivier Danvy. From reduction-based to reduction-free normalization. In Pieter W. M. Koopman, Rinus Plasmeijer, and S. Doaitse Swierstra, editors, *Advanced Functional Programming, 6th International School, AFP 2008, Heijen, The Netherlands, May 2008, Revised Lectures*, volume 5832 of *Lecture Notes in Computer Science*, pages 66–164. Springer, 2008.

27 Olivier Danvy and Lasse R. Nielsen. Syntactic theories in practice. *Electron. Notes Theor. Comput. Sci.*, 59(4):358–374, 2001.

28 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.

29 Nicholas G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae*, 34(5):381–392, 1972.

30 Marc Feeley. *An efficient and general implementation of futures on large scale shared-memory multiprocessors*. PhD thesis, Department of Computer Science, Brandeis University, 1993.

31 Matthias Felleisen, Robert Bruce Findler, and Matthew Flatt. *Semantics Engineering with PLT Redex*. The MIT Press, 2009.

32 Matthias Felleisen and Daniel P. Friedman. Control operators, the SECD-machine, and the λ-calculus. In Martin Wirsing, editor, *Formal Description of Programming Concepts - III: Proceedings of the IFIP TC 2/WG 2.2 Working Conference on Formal Description of Programming Concepts - III, Ebberup, Denmark, 25-28 August 1986*, pages 193–222. North-Holland, 1987.

33 Cormac Flanagan and Matthias Felleisen. The semantics of future and an application. *J. Funct. Program.*, 9(1):1–31, 1999. doi:10.1017/S0956796899003329.

34 Álvaro García-Pérez and Pablo Nogueira. The full-reducing Krivine abstract machine KN simulates pure normal-order reduction in lockstep: A proof via corresponding calculus. *J. Funct. Program.*, 29:e7, 2019. doi:10.1017/S0956796819000017.

35 Álvaro García-Pérez, Pablo Nogueira, and Juan José Moreno-Navarro. Deriving the full-reducing Krivine machine from the small-step operational semantics of normal order. In Ricardo Peña and Tom Schrijvers, editors, *15th International Symposium on Principles and Practice of Declarative Programming, PPDP '13, Madrid, Spain, September 16-18, 2013*, pages 85–96. ACM, 2013. doi:10.1145/2505879.2505887.

36 Simon L. Peyton Jones. Implementing lazy functional languages on stock hardware: The Spineless Tagless G-machine. *J. Funct. Program.*, 2(2):127–202, 1992.

37 David A. Kranz, Robert H. Halstead Jr., and Eric Mohr. Mul-t: A high-performance parallel Lisp. In Richard L. Wexelblat, editor, *Proceedings of the ACM SIGPLAN'89 Conference on Programming Language Design and Implementation (PLDI), Portland, Oregon, USA, June 21-23, 1989*, pages 81–90. ACM, 1989.

38 Jean-Louis Krivine. A call-by-name lambda-calculus machine. *Higher-Order and Symbolic Computation*, 20(3):199–207, 2007.

39 Peter J. Landin. The mechanical evaluation of expressions. *The Computer Journal*, 6(4):308–320, 1964.

40 Ivan Lanese, Jorge A. Pérez, Davide Sangiorgi, and Alan Schmitt. On the expressiveness and decidability of higher-order process calculi. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pages 145–155. IEEE Computer Society, 2008.

41 Xavier Leroy. The ZINC experiment: an economical implementation of the ML language. Technical report 117, INRIA, 1990.

42 Joachim Niehren, Jan Schwinghammer, and Gert Smolka. A concurrent lambda calculus with futures. *Theor. Comput. Sci.*, 364(3):338–356, 2006. doi:10.1016/J.TCS.2006.08.016.

43 Luca Paolini and Simona Ronchi Della Rocca. Call-by-value solvability. *RAIRO Theor. Informatics Appl.*, 33(6):507–534, 1999. doi:10.1051/ita:1999130.

44 Gordon D. Plotkin. A structural approach to operational semantics. Technical Report FN-19, DAIMI, Department of Computer Science, Aarhus University, Aarhus, Denmark, September 1981.

45 Kristoffer H. Rose. Explicit substitutions – tutorial & survey. BRICS Lecture Series LS-96-3, DAIMI, Aarhus, Denmark, September 1996.

46 Sylvain Salvati and Igor Walukiewicz. Krivine machines and higher-order schemes. *Inf. Comput.*, 239:340–355, 2014.

47 Davide Sangiorgi. Bisimulation in higher-order process calculi. In Ernst-Rüdiger Olderog, editor, *Programming Concepts, Methods and Calculi, Proceedings of the IFIP TC2/WG2.1/WG2.2/WG2.3 Working Conference on Programming Concepts, Methods and Calculi (PROCOMET '94) San Miniato, Italy, 6-10 June, 1994*, volume A-56 of *IFIP Transactions*, pages 207–224. North-Holland, 1994.

48 Peter Sestoft. Deriving a lazy abstract machine. *J. Funct. Program.*, 7(3):231–264, 1997. doi:10.1017/s0956796897002712.

49 C.P. Wadsworth. *Semantics and Pragmatics of the Lambda-Calculus*. PhD thesis, University of Oxford, 1971.

A Linear Type System for L^p-Metric Sensitivity Analysis

Victor Sannier
Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

Patrick Baillot
Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

Abstract

When working in optimisation or privacy protection, one may need to estimate the sensitivity of computer programs, i.e., the maximum multiplicative increase in the distance between two inputs and the corresponding two outputs. In particular, differential privacy is a rigorous and widely used notion of privacy that is closely related to sensitivity. Several type systems for sensitivity and differential privacy based on linear logic have been proposed in the literature, starting with the functional language Fuzz. However, they are either limited to certain metrics (L^1 and L^∞), and thus to the associated privacy mechanisms, or they rely on a complex notion of type contexts that does not interact well with operational semantics. We therefore propose a graded linear type system – inspired by Bunched Fuzz [27] – called Plurimetric Fuzz that handles L^p vector metrics (for $1 \leq p \leq +\infty$), uses standard type contexts, gives reasonable bounds on sensitivity, and has good metatheoretical properties. We also provide a denotational semantics in terms of metric complete partial orders, and translation mappings from and to Fuzz.

2012 ACM Subject Classification Theory of computation → Type theory; Theory of computation → Linear logic; Security and privacy → Logic and verification

Keywords and phrases type system, linear logic, sensitivity, vector metrics, differential privacy, lambda-calculus, functional programming, denotational semantics

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.12

Related Version *Full Version*: `https://hal.science/hal-04514677`

Acknowledgements The authors would like to thank Arthur Azevedo de Amorim for his valuable comments, in particular on denotational semantics.

1 Introduction

The sensitivity of a program is a measure of how much the result of the computation depends on its inputs, and is defined with respect to some metrics on data. Concretely, if d_X and d_Y are metrics on the input and output spaces respectively, and if f is the function computed by the program, its sensitivity is the smallest positive real s such that $d_Y\big(f(x), f(x')\big) \leq s \cdot d_X(x, x')$, for any pairs of inputs (x, x'). This notion is important for analysing the stability of some machine-learning algorithms, or the privacy properties of a program [9, 12]. In particular, sensitivity is a key notion for *differential privacy* [14, 15], a popular approach to the protection of sensitive data, like medical records, that provides mathematically-based, rigorous and composable guarantees. The intuition behind differential privacy is that one can hide the information about whether or not a given individual is included in the input dataset by perturbating the result of the function. In practice, one adds a well-calibrated amount of random noise to the result, by means of specific *mechanisms*: one should not be able to deduce from the output whether the individual belongs to the input or not, but the result should still be accurate enough.

As the analysis of sensitivity and the implementation of differential privacy are delicate and error-prone tasks, some approaches in the programming languages community have been developed to assist programmers. They can be categorised into two classes: those based on Hoare logics [6, 7, 5], which are interactive and suitable for verifying mechanism implementations, and those based on type systems [23, 16], which are automatisable and well-suited for verifying functional programs that compose mechanisms. Moreover recent works [21, 24] on type systems have suggested that the analysis of sensitivity and privacy in these systems could be handled essentially separately, by using two different classes of typing judgements.

In this paper, we are interested in the type systems for the analysis of sensitivity. The seminal work on the Fuzz language by Reed and Pierce [23] has shown how ideas from linear logic [18, 19] can be used to design a type system for a functional language which statically bounds the sensitivity of a program by providing connectives which can express two metrics on vectors: the L^1 and the L^∞ metrics. However depending on the applications some other metrics on vectors are relevant. For instance, for many geometric algorithms one is interested in the Euclidean distance L^2, and more generally, in the literature on optimisation and statistical applications [10, 20], L^p distances with $1 \leq p \leq +\infty$ have been used to advantage. For this reason, wunder et al. [27] have introduced an extension of Fuzz, called Bunched Fuzz, which features connectives allowing to handle L^p-metrics ($1 \leq p \leq +\infty$) on vectors. The derivations of this system use generalised typing judgements inspired by the logic of Bunched Implications [22], where typing contexts have a tree structure. The authors established a soundness result analogous to that of Fuzz, showing that the functions computed by well-typed programs admit a certain sensitivity property.

In the following, we will discuss why Bunched Fuzz does not satisfy the desired properties with respect to an operational semantics, and we will design a type system for L^p metrics inspired by Bunched Fuzz with the following expectations: (i) sensitivity soundness property, (ii) substitution and subject-reduction property, (iii) subtyping property, and (iv) expressiveness. Requirement (iii) refers to the fact that for all p and q, the L^p and L^q-metrics are related by two inequalities that can be used for coercions between data types convenient for composing functions. As to (iv), we mean that we want the system to be able to type some meaningful examples.

Concretely, we keep the same type language as Bunched Fuzz, but we consider a system of rules that uses standard judgements with list contexts, we call this system Plurimetric Fuzz. As an additional benefit, we will define (partial) translation mappings from Fuzz to Plurimetric Fuzz, and vice versa, that we think shed some light on how the new system refines Fuzz.

1.1 Summary of Contributions

We introduce Plurimetric Fuzz, a type system with recursive types and a form of subtyping (see Section 3.4) for bounding the L^p-sensitivity of vector-valued functions, which subsumes Fuzz ($p = 1$). We show that Plurimetric Fuzz enjoys the subject reduction property (Theorem 5.2), and that it is sound with respect to its denotational semantics (Theorem 4.15). We also show that it gives significantly lower bounds on sensitivity compared to a naïve extension of Fuzz, and that it is expressive enough to prove a classification algorithm $(\epsilon, 0)$-private (see Section 6).

2 Background

We first give an overview of the notions and results about sensitivity, differential privacy and type systems that will be needed in the paper.

2.1 Metric Spaces and Sensitivity

▶ **Definition 2.1.** *An* extended pseudosemimetric space*, or* metric space *for short, is a pair* (X, d) *where* X *is a set and* $d : X \times X \to [0, \infty]$ *is a function such that for all* $x, y, z \in X$*: (1)* $d(x, y) = 0$ *if* $x = y$*; and (2)* $d(x, y) = d(y, x)$*.*

Note that we do not require the triangle inequality to hold.

In this paper, we are interested in a family of metrics over $\mathbf{R}^d$, which are defined as follows, and related by the inequalities of Lemma 2.3.

▶ **Definition 2.2.** *For all parameter* $p \geq 1$ *and for all vectors* $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_d)$ *and* $\mathbf{y} = (\mathbf{y}_1, \ldots, \mathbf{y}_d)$ *in* $\mathbf{R}^d$*, we define the* L^p-distance *or the* vector metric of parameter p *between* $\mathbf{x}$ *and* $\mathbf{y}$ *by* $d_p(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{d} |\mathbf{x}_i - \mathbf{y}_i|^p\right)^{1/p}$.

▶ **Lemma 2.3.** *For all parameters* p *and* q *such that* $1 \leq p, q \leq \infty$*, let* $c(p, q) = 2^{|1/p - 1/q|}$*. If* $p \leq q$*, then we have* $d_p \geq d_q \geq c(p, q) \cdot d_p$*.*

The *sensitivity* of a map between metric spaces is a measure of how much its output changes when its input changes. This notion is useful for analysing the privacy guarantees of probabilistic algorithms, as we will see in Section 2.2.

▶ **Definition 2.4.** *A map* f *between two metric spaces* (X, d_X) *and* (Y, d_Y) *is said to be* s-sensitive*, or* s*-Lipschitz continuous, for* $s \in [0, \infty]$ *if for all points* x *and* x' *in* X*, we have* $d_Y\big(f(x), f(x')\big) \leq s \cdot d_X(x, x')$*. The* sensitivity *of* f *is the least real* s *such that* f *is* s*-sensitive. When it is bounded by* 1*, we say that* f *is* non-expansive*.*

▶ Remark 2.5. To perform operations on sensitivities, we extend addition to possibly infinite reals in a straightforward way and multiplication in the same way as [3, Section 2], that is such that $s \cdot \infty$ equals ∞, and $\infty \cdot s$ equals 0 if $s = 0$ and ∞ otherwise. Note that this operation is not commutative, see [24, Section 4.2] for a discussion on the soundness of this choice.

For differentiable real functions, sensitivity is related to the magnitude of the derivative.

▶ **Lemma 2.6.** *Let* f *be a differentiable function from* $\mathbf{R}$ *to* $\mathbf{R}$ *such that for all* $x \in \mathbf{R}$*, we have* $|f'(x)| \leq s$*. Then* f *is* s*-sensitive.*

2.2 Differential Privacy

A strong motivation for studying sensitivity lies in the field of privacy-preserving data analysis. Informally, differential privacy [14, 15] is a strong statistical notion of privacy, probably the most widely used and studied, which requires that the outcome of a computation should not depend too much on the presence or absence of a single record in the input database.

▶ **Definition 2.7.** *A probabilistic algorithm* A *endowed with an adjacency relation is said to be* (ϵ, δ)-differentially private *for some* $\epsilon \geq 0$ *and* $\delta \in [0, 1]$ *if, for all adjacent inputs* x *and* x'*, and all subsets* S *of* $\mathrm{codom}(A)$*, we have* $\Pr[A(x) \in S] \leq e^{\epsilon} \Pr[A(x') \in S] + \delta$*.*

▶ Remark. Differential privacy can also be defined in terms of a hypothesis-testing problem, where an adversary attempts to distinguish between two adjacent inputs by observing the outcome of the algorithm [25].

In practice, X will often be the set of databases, and two databases will be adjacent if one can be obtained from the other by adding or removing a single record. Moreover, the codomain will often be of the form $\mathbf{R}^d$ and endowed with a vector metric such as the Manhattan distance ($p = 1$) or the Euclidean distance ($p = 2$).

In order to guarantee differential privacy, it is enough to add noise to the computation, as long as the noise is sufficiently large compared to the sensitivity of the function being computed. Let us give more precise statements.

Let f be a vector-valued function from a metric space (X, d_X) to $\mathbf{R}^d$. We write $\Delta_p f$ for the sensitivity of f when the codomain is endowed with the L^p-distance, that is for the L^p-sensitivity of f. Recall that the *Laplace distribution* of parameter $b > 0$ is the probability distribution with density function $x \mapsto 1/2b \cdot e^{-|x|/b}$, for x in $\mathbf{R}$.

▶ **Theorem 2.8** (Laplace Mechanism [15, Theorem 3.6]). *If $\Delta_1 f$ is finite, then for all positive real number ϵ, the function $x \mapsto f(x) + \big(\mathrm{Lap}(\Delta_1 f/\epsilon), \dots, \mathrm{Lap}(\Delta_1 f/\epsilon)\big)$ is ϵ-differentially private.*

However, in some cases, we may prefer to add Gaussian noise instead of Laplace noise. This way, the noise added to protect privacy is of the same type as other sources of perturbation in the original data. Moreover, the effects of the privacy mechanism on the statistical analysis may be easier to account for given that the sum of normally distributed random variables is itself normally distributed [15, Section 3.5.3]. To do so, we need to bound the L^2-sensitivity of f.

▶ **Theorem 2.9** (Gaussian mechanism [15, Theorem 3.22]). *If $\Delta_2 f$ is finite, then for all positive real numbers ϵ and δ, if $\sigma > \sqrt{2\ln(5/4\delta)} \cdot \Delta_2 f/\epsilon$, then the function $x \mapsto f(x) + \big(\mathcal{N}(0,\sigma^2), \dots, \mathcal{N}(0,\sigma^2)\big)$ is (ϵ,δ)-differentially private.*

▶ Remark 2.10. In this paper, we will only consider discrete probability distributions, but the above two theorems can be adapted to this setting [17, 11].

2.3 Type Systems for Bounding Sensitivity

Reed and Pierce have introduced the Fuzz type system [23] based on the fact that L^1-sensitivity can be viewed as an affine resource (in the sense of linear logic [18, 19]). For example, the judgement $[x : A]_2 \vdash (x, x) : A \otimes A$ means that the map $x \mapsto (x, x)$ is 2-sensitive (for the L^1-distance). See the following tensor rules for an example of Fuzz typing rules:

$$\frac{\Gamma \vdash a : A \qquad \Delta \vdash b : B}{\Gamma + \Delta \vdash (a, b) : A \otimes B}\otimes I \qquad \frac{\Delta \vdash e : A \otimes B \qquad \Gamma, [x : A]_s, [y : B]_s \vdash c : C}{\Gamma + s\Delta \vdash (\mathbf{let}\ (x, y) = e\ \mathbf{in}\ c) : C}\otimes E$$

where the sum of two contexts is the result of adding the sensitivities of the involved variables.

It was subsumed by Bunched Fuzz [27], which allows for the analysis of L^p-sensitivity for $p \in [1, \infty]$ by the introduction of a family of tensor products $(\otimes_p)_{p\in[1,\infty]}$ and affine arrows $(\multimap_p)_{p\in[1,\infty]}$. Moreover contexts Γ are no longer represented as lists, but as trees (or *bunches*). We reproduce the tensor rules below:

$$\frac{\Gamma \vdash a : A \qquad \Delta \vdash b : B}{\Gamma \,,_p\, \Delta \vdash (a, b) : A \otimes_p B}\otimes I \qquad \frac{\Delta \vdash e : A \otimes_p B \qquad \Gamma\big([x : A]_s \,,_p\, [y : B]_s\big) \vdash c : C}{\Gamma(s\Delta) \vdash (\mathbf{let}\ (x, y) = e\ \mathbf{in}\ c) : C}\otimes E$$

where $\Gamma(\Delta)$ denotes a composite bunch formed by substituting the bunch Δ into another bunch $\Gamma(\star)$, which features a unique, distinguished hole $\star$.

▶ Remark 2.11. We write $b[a/x]$ for the capture-avoiding substitution of b for x in a.

The contraction rule enables the identification of variables with the same type in two different subtrees of a context,

$$\frac{\Gamma(\Delta \,,_p\, \Delta') \vdash a : A \qquad \Delta \approx \Delta'}{\Gamma\big(\mathrm{Contr}(p; \Delta; \Delta')\big) \vdash a[\mathrm{vars}\,\Delta/\,\mathrm{vars}\,\Delta'] : A}\ \mathrm{Contr}$$

where $\mathrm{Contr}(p;\Gamma;\Delta)$ is defined by induction on the structure of Γ by the following equations, and where we write $\cdot$ for the sensitivity scaling operation.

$$\begin{aligned}
\mathrm{Contr}(p;\emptyset;\emptyset) &\stackrel{\text{def}}{=} \emptyset \\
\mathrm{Contr}(p;[x:A]_r;[y:A]_s) &\stackrel{\text{def}}{=} [x:A]_{\sqrt[p]{r^p+s^p}} \\
\mathrm{Contr}(p;\Gamma_1 \,,_q\, \Gamma_2;\Delta_1 \,,_q\, \Delta_2) &\stackrel{\text{def}}{=} 2^{|1/p-1/q|}\cdot\big(\mathrm{Contr}(p;\Gamma_1;\Delta_1) \,,_q\, \mathrm{Contr}(p;\Gamma_2;\Delta_2)\big)
\end{aligned}$$

The authors have proved that if types and contexts are interpreted as metric spaces, then the derivations correspond to non-expansive functions.

▶ **Theorem 2.12** ([27, Theorem 7]). *Given a derivation π proving $\Gamma \vdash a : A$, the function $[\![\pi]\!] : [\![\Gamma]\!] \to [\![A]\!]$ is non-expansive.*

However, the use of bunches comes at the cost of the loss of the substitution property: there exists derivations $\Gamma \vdash a : A$ and $\Delta([x:A]_s) \vdash b : B$ such that $\Delta(s\Gamma) \not\vdash b[a/x] : B$.

Proof. To see this, let us look at the following example. Here, we use the algorithmic approach to the rules, which means we systematically apply a contraction after each typing rule:

$$\dfrac{\dfrac{}{\emptyset \vdash (+) : \mathsf{Nat} \otimes_1 \mathsf{Nat} \multimap_1 \mathsf{Nat}}\ + \quad \dfrac{\dfrac{}{[a:\mathsf{Nat}]_1 \vdash a : \mathsf{Nat}}\ \text{var} \qquad \dfrac{}{[b:\mathsf{Nat}]_1 \vdash b : \mathsf{Nat}}\ \text{var}}{[a:\mathsf{Nat}]_1 \,,_1\, [b:\mathsf{Nat}]_1 \vdash (a,b) : \mathsf{Nat} \otimes_1 \mathsf{Nat}}\ \otimes I}{[a:\mathsf{Nat}]_1 \,,_1\, [b:\mathsf{Nat}]_1 \vdash (+)(a,b) : \mathsf{Nat}}\ \multimap E$$

If we could substitute $(+)(a,b)$ for x in the derivation

$$\dfrac{\dfrac{}{[x:\mathsf{Nat}]_1 \vdash x : \mathsf{Nat}}\ \text{var} \qquad \dfrac{}{[x:\mathsf{Nat}]_1 \vdash x : \mathsf{Nat}}\ \text{var}}{[x:\mathsf{Nat}]_{\sqrt{2}} \vdash (x,x) : \mathsf{Nat} \otimes_2 \mathsf{Nat}}\ \otimes I$$

we would obtain $[a:\mathsf{Nat}]_{\sqrt{2}} \,,_1\, [b:\mathsf{Nat}]_{\sqrt{2}} \vdash \big((+)(a,b),(+)(a,b)\big) : \mathsf{Nat} \otimes_2 \mathsf{Nat}$, which is not derivable. Indeed, a derivation of this judgement would have the following shape:

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ [a:\mathsf{Nat}]_{r_a} \,,_1\, [b:\mathsf{Nat}]_{r_b} \vdash (+)(a,b) : \mathsf{Nat}\end{array} \qquad \begin{array}{c}\vdots\\ [a:\mathsf{Nat}]_{s_a} \,,_1\, [b:\mathsf{Nat}]_{s_b} \vdash (+)(a,b) : \mathsf{Nat}\end{array}}{c(2,1)\cdot\big([a:\mathsf{Nat}]_1 \,,_1\, [b:\mathsf{Nat}]_1\big) \vdash \big((+)(a,b),(+)(a,b)\big) : \mathsf{Nat} \otimes_2 \mathsf{Nat}}\ \otimes I}{[a:\mathsf{Nat}]_{\sqrt{2}} \,,_1\, [b:\mathsf{Nat}]_{\sqrt{2}} \vdash \big((+)(a,b),(+)(a,b)\big) : \mathsf{Nat} \otimes_2 \mathsf{Nat}}\ =$$

where r_a, r_b, s_a and s_b would be such that $r_a^2 + s_a^2 = 1$ and $r_b^2 + s_b^2 = 1$. We would have $\min\{r_a, s_a\} \leq \sqrt{2}/2 < 1$, which is absurd as $(a,b) \mapsto a+b$ is 1-sensitive for the L^1-metric. ◀

▷ Claim 2.13. Bunched Fuzz doesn't meet the subject reduction property when it is given a standard operational semantics similar to that of Fuzz (see Figure 3).

In addition, the failure to satisfy the substitution property implies that we cannot meaningfully state certain properties regarding denotational semantics (see Section 5 for the metatheoretical properties our type system enjoys). This includes the assertion, using the notations above, that $[\![b[a/x]]\!]$ is equal to $[\![b]\!]$ when partially applied to a, as the first term may not have a valid derivation, and therefore a well-defined interpretation.

In conclusion, the flexibility provided by representing contexts as trees is offset by the loss of important syntactic and semantic properties.

3 Syntax

In a nutshell we will consider the terms of Fuzz, that is to say an extended λ-calculus, with the types of Bunched Fuzz, but with a new notion of typing context.

3.1 Types and Terms

Types are defined by the following context-free grammar where s and p range over $[0,\infty]$ and $[1,\infty]$ respectively: $A, B, \dots ::= \mathsf{Unit} \mid A \oplus B \mid \mu\alpha.\ A \mid \bigcirc A \mid !_s A \mid A \otimes_p B \mid A \multimap_p B$. We write Bool for the type $\mathsf{Unit} \oplus \mathsf{Unit}$; $\mathsf{List}_p(A)$ for the iso-recursive type $\mu\alpha.\ \mathsf{Unit} \oplus (A \otimes_p \alpha)$; and $\bigotimes_p^d A = \overbrace{A \otimes_p \dots \otimes_p A}^{d \text{ times}}$.

On the other hand, the terms of the language are defined by the following grammar, for $\mathsf{c} \in \mathrm{Const}$, $x, y \in \mathrm{Var}$ and $A \in \mathrm{Typ}$:

$$\begin{aligned} a, b, c, d, e, f, \dots ::= {} & * \mid \mathsf{c} \mid x \mid (a, b) \mid \mathbf{let}\ (x, y) = e\ \mathbf{in}\ b \mid \pi_i e \mid \lambda x.\ e \mid f\ e \\ & \mid \mathbf{inj}_1\, e \mid \mathbf{inj}_2\, e \mid \mathbf{case}\ e\ \mathbf{of}\ x \Rightarrow a\ \mathbf{or}\ y \Rightarrow b \mid !e \mid \mathbf{let}\ !x = e\ \mathbf{in}\ b \\ & \mid \underset{A}{\mathbf{fold}}\, e \mid \underset{A}{\mathbf{unfold}}\, e \mid \mathbf{return}\, e \mid \mathbf{let}\ \bigcirc x = e\ \mathbf{in}\ b \end{aligned} \tag{1}$$

▶ Remark 3.1. In examples, we write terms in an ML-like syntax instead of the one described in Section 3. In particular, we may write `x |> f` for `f x`, we may use pattern matching and `let` bindings ($\mathbf{let}\ x = e\ \mathbf{in}\ b$ is syntactic sugar for $(\lambda x.\ b)\ e$), and we may omit the Y combinator (see Remark 3.5) when defining recursive functions.

3.2 Precontexts and Contexts

We refer to elements of the set defined by the grammar $\Gamma ::= \emptyset \mid [x : A]_s, \Gamma$ – where s, x and A range over $[0,\infty]$, Var and Type respectively – as *precontexts*. In addition, we define the scaling $s\Gamma$ of a precontext Γ by a sensitivity s by $s \cdot \emptyset \stackrel{\text{def}}{=} \emptyset$ and $s \cdot ([x : A]_r, \Gamma) \stackrel{\text{def}}{=} [x : A]_{rs}, s\Gamma$ for all $s \in [0,\infty]$.

▶ **Definition 3.2.** *Two precontexts Γ and Δ are said to be* compatible *if they do not assign different types to the same variable. The* p-contraction *of two compatible precontexts Γ and Δ is defined by induction on the structure of Γ by the following equations:*

$$\begin{aligned} C^p(\emptyset; \Delta) &\stackrel{def}{=} \Delta \\ C^p([x : A]_r, \Gamma; \Delta) &\stackrel{def}{=} [x : A]_r, C^p(\Gamma; \Delta) \qquad \text{if } x \notin \Delta \\ C^p([x : A]_r, \Gamma; [x : A]_s, \Delta) &\stackrel{def}{=} [x : A]_{\sqrt[p]{r^p + s^p}}, C^p(\Gamma; \Delta) \end{aligned} \tag{2}$$

We write $\Gamma + \Delta$ for $C^1(\Gamma; \Delta)$. ⌟

▶ **Lemma 3.3.** *For all precontexts Γ and Δ, and all parameters p:*

- $C^p(\Gamma; \Delta) = C^p(\Delta; \Gamma)$;
- *if* $C^p(\Gamma; \Delta) = \emptyset$, *then* $\Gamma = \emptyset$ *and* $\Delta = \emptyset$;
- *for all sensitivity s, we have* $s \cdot C^p(\Gamma; \Delta) = C^p(s\Gamma; s\Delta)$.

▶ **Definition 3.4.** *For any two precontexts Γ and Δ, we write $\Gamma \leq \Delta$ if every variable of Γ also occurs in Δ, and with greater or equal sensitivity.*

Finally, we define a *context* as a pair of a parameter and a precontext, which can be seen as a Bunched Fuzz context where all parameters are equal, and which can therefore be flattened into a list. More precisely, a *context* is a pair (p, Γ) written $(p)\ \Gamma$.

$$\frac{}{(p)\ [x : A]_1 \vdash x : A}\ \text{var} \qquad \frac{}{\vdash * : \mathsf{Unit}}\ \mathsf{Unit}$$

$$\frac{(p)\ \Gamma \vdash a : A \qquad (p)\ \Delta \vdash b : B}{(p)\ C^p\,(\Gamma;\Delta) \vdash (a,b) : A \otimes_p B}\ \otimes I$$

$$\frac{(p)\ \Gamma \vdash e : A \otimes_p B \qquad (p)\ \Delta, [x : A]_s, [y : B]_s \vdash c : C}{(p)\ C^p\,(s\Gamma;\Delta) \vdash (\mathbf{let}\ (x,y) = e\ \mathbf{in}\ c) : C}\ \otimes E$$

$$\frac{(p)\ \Gamma \vdash a : A}{(p)\ \Gamma \vdash \mathbf{inj}_1\, a : A \oplus B}\ \oplus I_\triangleleft \qquad \frac{(p)\ \Gamma \vdash b : B}{(p)\ \Gamma \vdash \mathbf{inj}_2\, b : A \oplus B}\ \oplus I_\triangleright$$

$$\frac{(p)\ \Gamma \vdash e : A \oplus B \qquad (p)\ \Delta, [x : A]_s \vdash c_1 : C \qquad (p)\ \Delta, [y : B]_s \vdash c_2 : C}{(p)\ C^p\,(s\Gamma;\Delta) \vdash (\mathbf{case}\ e\ \mathbf{of}\ x \Rightarrow c_1\ \mathbf{or}\ y \Rightarrow c_2) : C}\ \oplus E$$

$$\frac{(p)\ \Gamma \vdash a : A}{(p)\ s\Gamma \vdash\ !a : !_s A}\ !I \qquad \frac{(p)\ \Gamma \vdash e : !_r A \qquad (p)\ \Delta, [x : A]_{rs} \vdash c : C}{(p)\ C^p\,(s\Gamma;\Delta) \vdash (\mathbf{let}\ x = e\ \mathbf{in}\ c) : C}\ !E$$

$$\frac{(p)\ \Gamma, [x : A]_1 \vdash b : B}{(p)\ \Gamma \vdash (\lambda x.\ b) : A \multimap_p B}\ \multimap I \qquad \frac{(p)\ \Gamma \vdash f : A \multimap_p B \qquad (p)\ \Delta \vdash a : A}{(p)\ C^p\,(\Gamma;\Delta) \vdash f\ a : B}\ \multimap E$$

$$\frac{(p)\ \Gamma \vdash e : A[\mu\alpha.\ A/\alpha]}{(p)\ \Gamma \vdash \underset{\mu\alpha.\ A}{\mathbf{fold}}\, e : A}\ \mu I \qquad \frac{(p)\ \Gamma \vdash a : A}{(p)\ \Gamma \vdash \underset{\mu\alpha.\ A}{\mathbf{unfold}}\, a : A[\mu\alpha.\ A/\alpha]}\ \mu E$$

$$\frac{(1)\ \Gamma \vdash a : A}{(1)\ \infty \cdot \Gamma \vdash \mathbf{return}\, a : \bigcirc A}\ \bigcirc I \qquad \frac{(1)\ \Gamma \vdash e : \bigcirc A \qquad (1)\ \Delta, [x : A]_\infty \vdash b : \bigcirc B}{(1)\ \Gamma + \Delta \vdash (\mathbf{let}\ \bigcirc x = e\ \mathbf{in}\ b) : \bigcirc B}\ \bigcirc E$$

$$\frac{(p)\ \Gamma \vdash a : A \qquad \Gamma \leq \Delta \qquad p \geq q}{(q)\ \Delta \vdash a : A}\ \geq W \qquad \frac{(p)\ \Gamma \vdash a : A \qquad \Gamma \leq \Delta \qquad p \leq q}{(q)\ c(p,q) \cdot \Delta \vdash a : A}\ \leq W$$

Figure 1 Typing Rules for Plurimetric Fuzz.

3.3 Typing Rules

The typing rules and typing rules schemas for Plurimetric Fuzz are given in Figure 1 where Γ and Δ range over contexts, A, B, and C range over types, etc.

We omit the $\to_p$ type constructor, and encode it with $\multimap_p$ and $!_\infty$ as follows: $A \to_p B \stackrel{\text{def}}{=} !_\infty A \multimap_p B$. Similarly, the & constructor can be encoded by $\otimes_\infty$ like in [27, Section 3]. Observe that, as $C^1\,(\Gamma;\Delta) = \Gamma + \Delta$, all rules but the last two, $(\geq W)$ and $(\leq W)$, correspond to Fuzz rules when $p = 1$ (by identifying connectives of parameter 1 with the corresponding Fuzz ones). So all Fuzz type derivations can be seen as Plurimetric Fuzz type derivations (up to the encoding of &). We will see in Section 7 other ways of translating Fuzz derivations, by choosing other values of p.

Also note that the weakening rules $(\geq W)$ and $(\leq W)$ are the only ones that make the parameter of the judgement change. One direction $(\geq W)$ is direct, but the other one $(\leq W)$ requires a coefficient $c(p,q) = 2^{|1/p - 1/q|}$ (see Lemma 2.3). In addition, as a particular case of these rules, for all parameters p and q, from $(p)\ \emptyset \vdash a : A$ we can derive $(q)\ \emptyset \vdash a : A$. For this reason, we may simply write $\vdash a : A$.

$$\frac{}{\vdash n : \mathsf{Nat}}\ \mathsf{Nat} \qquad \frac{}{\vdash r : \mathsf{Real}}\ \mathsf{Real}$$

$$\frac{(p)\ \Gamma \vdash x : N \qquad (p)\ \Delta \vdash y : N \qquad N \in \{\mathsf{Nat}, \mathsf{Real}\}}{(p)\ c(1,p) \cdot C^p\,(\Gamma; \Delta) \vdash x + y : N}\ +$$

$$\frac{(p)\ \Gamma \vdash x : N \qquad k \in N \qquad N \in \{\mathsf{Nat}, \mathsf{Real}\}}{(p)\ k\Gamma \vdash k \times x : N}\ \times$$

$$\frac{(p)\ \Gamma \vdash x : A}{(p)\ \infty \cdot \Gamma \vdash \{x\} : \mathsf{Set}(A)}\ \mathsf{Set} \qquad \frac{(p)\ \Gamma \vdash e : \mathsf{Set}(A)}{(p)\ \Gamma \vdash \mathrm{card}(e) : \mathsf{Nat} \oplus \mathsf{Unit}}\ \mathrm{card}$$

$$\frac{}{\vdash \mathrm{setfilter} : (A \to_p \mathsf{Bool}) \to_p \mathsf{Set}(A) \multimap_p \mathsf{Set}(A)}\ \mathrm{setfilter}$$

$$\frac{}{\vdash \mathrm{setmap} : (A \to_p B) \to_p \mathsf{Set}(A) \multimap_p \mathsf{Set}(B)}\ \mathrm{setmap}$$

$$\frac{}{\vdash \mathrm{setfold} : (A \multimap_p B \multimap_p B) \to_p B \to_p \mathsf{Set}(A) \multimap_p B}\ \mathrm{setfold}$$

Figure 2 Typing Rules for Primitive Operations.

▶ Remark 3.5. As shown in [23, Section 3.1], recursive types let us encode a fix-point combinator for any two types A and B, and parameters p without a specific rule:

$$Y \stackrel{\text{def}}{=} \lambda f.\ \Big(\lambda x.\ \lambda a.\ f\big((\underset{A_0}{\mathbf{unfold}}\ x)x\big)a\Big)\Big(\underset{A_0}{\mathbf{fold}}\Big(\lambda x.\ \lambda a.\ f\big((\underset{A_0}{\mathbf{unfold}}\ x)x\big)a\Big)\Big)$$

where $A_0 \stackrel{\text{def}}{=} \mu\alpha\ \big(\alpha \to_p (A \multimap_p B)\big)$, and $Y : \big((A \multimap_p B) \to_p (A \multimap_p B)\big) \to_p (A \multimap_p B)$. ⌟

We can extend the type system to handle primitive operations on natural and real numbers, as well as on sets, having extending the syntax of types and terms accordingly in Figure 2.

3.4 Subtyping

For all p and q, the L^p and L^q-metrics are related by two inequalities that can be used for coercions between data types: if $p \leq q$, from $(p)\ \Gamma \vdash e : A \otimes_p B$, we can derive $(p)\ \Gamma \vdash \big(\mathbf{let}\ (x,y) = e\ \mathbf{in}\ (x,y)\big) : A \otimes_q B$; and similarly from $(q)\ \Gamma \vdash e : A \otimes_q B$, we can derive $(q)\ c(p,q) \cdot \Gamma \vdash \big(\mathbf{let}\ (x,y) = e\ \mathbf{in}\ (x,y)\big) : A \otimes_p B$. Let us give the derivation of the first case:

$$\dfrac{(p)\ \Gamma \vdash e : A \otimes_p B \qquad \dfrac{\dfrac{\dfrac{}{(q)\ [x:A]_1 \vdash x : A}\ \mathrm{var} \qquad \dfrac{}{(q)\ [y:B]_1 \vdash y : B}\ \mathrm{var}}{(q)\ [x:A]_1, [y:B]_1 \vdash (x,y) : A \otimes_q B}\ \otimes I}{(p)\ [x:A]_1, [y:B]_1 \vdash (x,y) : A \otimes_q B}\ \geq W}{(p)\ \Gamma \vdash \big(\mathbf{let}\ (x,y) = e\ \mathbf{in}\ (x,y)\big) : A \otimes_q B}\ \otimes E$$

▶ **Example 3.6.** As an example, say we want to compose a function $f\colon \mathsf{Real} \multimap_1 \mathsf{Real} \otimes_2 \mathsf{Real}$ with a function $g\colon \mathsf{Real} \otimes_1 \mathsf{Real} \multimap_1 \mathsf{Real}$, both typable in an empty context. We can first apply f to an input x, and then coerce the result to obtain the judgement $(1)\ [x : \mathsf{Real}]_{\sqrt{2}} \vdash e : \mathsf{Real} \otimes_1 \mathsf{Real}$ for some term e which is semantically equivalent to the term $f(x)$. At this point, we can apply g to e, and use the $(!I)$ and $(\multimap\ I)$ rules to derive the judgement $(1)\ \emptyset \vdash h : !_{\sqrt{2}}\mathsf{Real} \multimap_1 \mathsf{Real}$ for some term h which behaves like $g \circ f$.

$$\frac{}{* \Downarrow *} \qquad \frac{}{\lambda x.\, b \Downarrow \lambda x.\, b} \qquad \frac{f \Downarrow \lambda x.\, b \qquad a \Downarrow v_a \qquad b[v_a/x] \Downarrow v}{f\ a \Downarrow v}$$

$$\frac{a \Downarrow v_a \qquad b \Downarrow v_b}{(a,b) \Downarrow (v_a, v_b)} \qquad \frac{c \Downarrow (v_a, v_b) \qquad e[v_a/x][v_b/y] \Downarrow v}{(\mathbf{let}\ (x,y) = c\ \mathbf{in}\ e) \Downarrow v}$$

$$\frac{e \Downarrow v}{\mathbf{inj}_i\, e \Downarrow \mathbf{inj}_i\, v} \qquad \frac{e \Downarrow \mathbf{inj}_i\, v \qquad e_i[v/x] \Downarrow v_i}{(\mathbf{case}\ e\ \mathbf{of}\ x \Rightarrow e_1\ \mathbf{or}\ x \Rightarrow e_2) \Downarrow v_i}$$

$$\frac{e \Downarrow v}{!e \Downarrow !v} \qquad \frac{b \Downarrow\, !v_b \qquad e[v_b/x] \Downarrow v}{(\mathbf{let}\ !x = b\ \mathbf{in}\ e) \Downarrow v} \qquad \frac{e \Downarrow v}{\mathbf{fold}_A\, e \Downarrow \mathbf{fold}_A\, v} \qquad \frac{e \Downarrow \mathbf{fold}_A\, v}{\mathbf{unfold}_A\, e \Downarrow v}$$

$$\frac{e \Downarrow v}{\mathbf{return}\, e \Downarrow (1, v)} \qquad \frac{d \Downarrow (p_i, v_i)_{i \in I} \qquad \dfrac{i \in I}{b[v_i/x] \Downarrow (q_{ij}, w_{ij})_{j \in J}}}{\mathbf{let}\ \bigcirc x = d\ \mathbf{in}\ b \Downarrow (p_i q_{ij}, w_{ij})_{i \in I, j \in J}}$$

Figure 3 Evaluation rules for (Plurimetric) Fuzz.

4 Semantics

4.1 Operational Semantics

We consider the same big-step operational semantics as for Fuzz [23]. First, values are given by the following grammar: $u, v, \dots ::= * \mid (u, v) \mid \lambda x.\, b \mid\, !v \mid \mu \mid \mathbf{fold}_A\, v \mid \mathbf{inj}_1\, v \mid \mathbf{inj}_2\, v$ where μ ranges over multisets of probability-value pairs.

See Figure 3 for the complete set of evaluation rules, which can be extended with rules for primitive operations. We will see in the Section 5 that this semantics enjoys the desired properties such as subject reduction (also known as type preservation).

4.2 Denotational semantics

We also introduce a denotational semantics by interpreting types as metric spaces and type derivations as non-expansive maps, following the denotational semantics of Bunched Fuzz [27].

Operations on metric spaces

▶ **Definition 4.1.** *Let (X, d_X), (Y, d_Y) and (Z, d_Z) be three metric spaces, p be a parameter, and s be a sensitivity. The* scaling *of (X, d) by s is the metric space $!_s X \stackrel{def}{=} (X, s \cdot d_X)$. Moreover, the* p-tensor product *$X \otimes_p Y$ of X and Y is the set $X \times Y$ endowed with*

$$d_{X \otimes_p Y}\big((x,y),(x',y')\big) \stackrel{def}{=} \sqrt[p]{d_X(x,x')^p + d_Y(y,y')^p}\,; \tag{3}$$

the p-affine arrow *$X \multimap_p Y$ from X to Y is the set Y^X endowed with*

$$d_{X \multimap_p Y}(f, f') \stackrel{def}{=} \inf \left\{ r \geq 0 : \forall x, x' \in X, d_Y\big(f(x), f'(x')\big)^p \leq r^p + d_X(x,x')^p \right\}; \tag{4}$$

and the disjoint union *$X \oplus Y$ of X and Y is the set $X \sqcup Y$ endowed with*

$$d_{A_1 \oplus A_2}(e, e') \stackrel{def}{=} \begin{cases} d_i(e,e') & \text{if } e, e' \in [\![A_i]\!] \\ \infty & \text{otherwise.} \end{cases} \tag{5}$$

Given two maps $f\colon X \to Z$ and $g\colon Y \to Z$, the coproduct *$[f,g]\colon X \oplus Y \to Z$ of f and g is the map defined by $[f,g]\big(i_1(x)\big) \stackrel{def}{=} f(x)$, and $[f,g]\big(i_2(y)\big) \stackrel{def}{=} g(y)$.* ⌟

Note that it follows directly from the definitions above that for all p, the operation $\otimes_p$ is commutative and associative up to isomorphism, and that the evaluation map $\mathrm{Ev}\colon (X \multimap_p Y) \otimes_p X \multimap_p Y$ is non-expansive. Moreover, we have $(X \oplus Y) \otimes_p Z \simeq (X \otimes_p Z) \oplus (Y \otimes_p Z)$.

We also define probability distributions over metric spaces.

▶ **Definition 4.2.** *A* discrete probability distribution *over a metric space with countable support X is a function $\mu : X \to [0,1]$ such that $\sum_{x \in X} \mu(x) = 1$. We write $\mathrm{Dist}(X)$ for the set of such distributions endowed with the following distance, parametrised by a positive real ϵ_0:*

$$\max \mathrm{div}(\mu, \mu') \stackrel{def}{=} \frac{1}{\epsilon_0} \max_{x \in X} \left| \ln \frac{\mu(x)}{\mu'(x)} \right| \tag{6}$$

with the convention that $0/0 \stackrel{def}{=} 1$ and $|\ln(0/x)| \stackrel{def}{=} |\ln(x/0)| \stackrel{def}{=} \infty$ for all $x > 0$. We write δ_x for the Dirac distribution at x: $\delta_x(x) = 1$ and $\delta_x(x') = 0$ for all $x' \neq x$. ⌟

Recall that the *support* $\operatorname{supp} \mu$ of a distribution μ over a set X is the set of elements of X with non-zero probability, and note that if μ and μ' are two discrete distributions over the same set X, then $\max \mathrm{div}(\mu, \mu')$ is finite if and only if $\operatorname{supp} \mu = \operatorname{supp} \mu'$. This distance is "carefully chosen" [23, Section 4.2] to ensure that the following lemma holds.

▶ **Lemma 4.3.** *A non-expansive map from X to $\mathrm{Dist}(Y)$ is exactly an ϵ_0-differentially private random map from X to Y.*

To compose probabilistic programs, we define the Kleisli extension of a map.

▶ **Definition 4.4.** *The* Kleisli extension $f^\dagger : \mathrm{Dist}(X) \to \mathrm{Dist}(Y)$ *of a map $f : X \to \mathrm{Dist}(Y)$ is defined by the following formula: $f(\mu)(y) \stackrel{def}{=} \sum_{x \in X} \mu(x) f(x)(y)$.*

Interpretation of Types, Contexts and Derivations

Let Core Plurimetric Fuzz be the fragment of Plurimetric Fuzz without recursive types. We interpret its types inductively as metric spaces:

- $[\![\mathsf{Unit}]\!] \stackrel{\text{def}}{=} (\{*\}, 0)$;
- $[\![\mathsf{Nat}]\!] \stackrel{\text{def}}{=} (\mathbf{N}, (m,n) \mapsto |m-n|)$;
- $[\![\mathsf{Real}]\!] \stackrel{\text{def}}{=} (\mathbf{R}, (x,y) \mapsto |x-y|)$;
- $[\![!_s A]\!] \stackrel{\text{def}}{=} !_s [\![A]\!]$;
- $[\![A \otimes_p B]\!] \stackrel{\text{def}}{=} [\![A]\!] \otimes_p [\![B]\!]$;
- $[\![A \multimap_p B]\!] \stackrel{\text{def}}{=} [\![A]\!] \multimap_p [\![B]\!]$;
- $[\![A \oplus B]\!] \stackrel{\text{def}}{=} [\![A]\!] \oplus [\![B]\!]$;
- $[\![\bigcirc A]\!] \stackrel{\text{def}}{=} \mathrm{Dist}\, [\![A]\!]$;
- $[\![\mathsf{Set}(A)]\!] \stackrel{\text{def}}{=} \big(\mathcal{P}_{\text{finite}}([\![A]\!]), \mathrm{card}(- \mathbin{\triangle} -)\big)$.

where $\triangle$ is the symmetric difference on sets. Next, we define the interpretation of contexts: $[\![(p)\ \emptyset]\!] \stackrel{\text{def}}{=} \{*\}$ and $[\![(p)\ \Gamma, x : A]\!] \stackrel{\text{def}}{=} [\![(p)\ \Gamma]\!] \otimes_p [\![A]\!]$. Derivations are seen as non-expansive maps between metric spaces. More precisely, if π is a derivation whose last rule is R, we write π_e for its premise whose conclusion has term e, and Γ and Δ for the contexts involved. Moreover, we write $\widehat{-}$ for the currying map; Ev for the evaluation map; if R is a weakening rule, I_p for the inclusion map from $[\![(p)\ \Delta]\!]$ to $[\![(p)\ \Gamma]\!]$ when $\Gamma \leq \Delta$, and I_p^q for the natural map from $[\![(p)\ \Gamma]\!]$ to $[\![(q)\ \Gamma]\!]$; and if R is binary or ternary, D_p for the diagonal-like map from $[\![(p)\ C^p\,(\Gamma; \Delta)]\!]$ to $[\![(p)\ \Gamma]\!] \otimes_p [\![(p)\ \Delta]\!]$. We omit isomorphisms when they are clear from the context.

(Unit) $[\![\pi]\!] \stackrel{\text{def}}{=} \mathrm{Const}_*$

(var) $[\![\pi]\!] \stackrel{\text{def}}{=} \mathrm{Id}_{[\![A]\!]}$

($\otimes I$) $[\![\pi]\!] \stackrel{\text{def}}{=} \big([\![\pi_a]\!] \times [\![\pi_b]\!]\big) \circ D_p$

($\otimes E$) $[\![\pi]\!] \stackrel{\text{def}}{=} [\![\pi_c]\!] \circ (\mathrm{Id} \times r \cdot [\![\pi_e]\!]) \circ D_p$

($\multimap I$) $[\![\pi]\!] \stackrel{\text{def}}{=} \widehat{[\![\pi_b]\!]}$

($\multimap E$) $[\![\pi]\!] \stackrel{\text{def}}{=} \mathrm{Ev} \circ \big([\![\pi_f]\!] \times [\![\pi_a]\!]\big) \circ D_p$

$(\oplus I_\triangleleft)$ $[\![\pi]\!] \stackrel{\text{def}}{=} i_1 \circ [\![\pi_a]\!]$
$(\oplus I_\triangleright)$ $[\![\pi]\!] \stackrel{\text{def}}{=} i_2 \circ [\![\pi_b]\!]$
$(\oplus E)$ $[\![\pi]\!] \stackrel{\text{def}}{=} [[\![\pi_{c_1}]\!], [\![\pi_{c_2}]\!]] \circ (\mathrm{Id} \times s \cdot [\![\pi_e]\!]) \circ D_p$
$(!I)$ $[\![\pi]\!] \stackrel{\text{def}}{=} s \cdot [\![\pi_a]\!]$
$(!E)$ $[\![\pi]\!] \stackrel{\text{def}}{=} [\![\pi_c]\!] \circ \big(\mathrm{Id} \times r \cdot [\![\pi_e]\!]\big) \circ D_p$

$(\bigcirc I)$ $[\![\pi]\!] \stackrel{\text{def}}{=} \delta \circ \infty \cdot [\![\pi_a]\!]$
$(\bigcirc E)$ $[\![\pi]\!] \stackrel{\text{def}}{=} \mathrm{Ev} \circ \big((\infty \cdot [\![\pi_e]\!]) \times (\cdot^\dagger \circ \widehat{[\![\pi_b]\!]})\big) \circ D_1$
$(\leq W)$ $[\![\pi]\!] \stackrel{\text{def}}{=} c(p,q) \cdot I_q^p \circ I_q \circ [\![\pi_a]\!]$
$(\geq W)$ $[\![\pi]\!] \stackrel{\text{def}}{=} I_q^p \circ I_q \circ [\![\pi_a]\!]$

Soundness of Core Plurimetric Fuzz

Let Core Plurimetric Fuzz be the fragment of Plurimetric Fuzz without recursive types.

▶ **Proposition 4.5** (Soundness). *If π is a Core Plurimetric Fuzz derivation of $(p)\ \Gamma \vdash a : A$, then $[\![\pi]\!]$ is a non-expansive map from $[\![(p)\ \Gamma]\!]$ to $[\![A]\!]$.*

Properties we use repeatedly in the proof of this result are summarised in the following lemmata. See Appendix A.1 for more details.

▶ **Lemma 4.6.** *For all precontexts Γ, and reals $p \geq 1$ and $s \geq 0$, we have $[\![(p)\ s\Gamma]\!] = !_s[\![(p)\ \Gamma]\!]$.*

▶ **Lemma 4.7.** *For all metric spaces X_1, X_2, Y_1, and Y_2, and parameters p, if $f\colon X_1 \to Y_1$ and $g\colon X_2 \to Y_2$ are non-expansive maps, then so is $f \times g\colon X_1 \otimes_p X_2 \to Y_1 \otimes_p Y_2$.*

Proof. The function $(x,y) \mapsto \sqrt[p]{x^p + y^p}$ is increasing in both arguments over $\mathbf{R}_{\geq 0} \times \mathbf{R}_{\geq 0}$. ◀

▶ **Lemma 4.8** ([27, Proposition 6]). *For all metric spaces X and Y, and parameters p and q such that $p \leq q$, the identity map on pairs belongs to the following spaces: $X \otimes_p Y \multimap X \otimes_q Y$ and $!_{c(p,q)}(X \otimes_q Y) \multimap X \otimes_p Y$.*

▶ **Lemma 4.9.** *For all compatible precontexts Γ and Δ, the diagonal-like map D_p from $[\![(p)\ C^p\,(\Gamma;\Delta)]\!]$ to $[\![(p)\ \Gamma]\!] \otimes_p [\![(p)\ \Delta]\!]$ is non-expansive. Moreover, if $\Gamma \leq \Delta$, then the inclusion map $I_p\colon [\![(p)\ \Gamma]\!] \to [\![(p)\ \Delta]\!]$ is non-expansive.*

Proof. By induction on Γ, using the fact that for all metric spaces X, and sensitivities r and s, we have a non-expansive map from $!_{\sqrt[p]{r^p+s^p}}X$ to $!_r X \otimes_p !_s X$ given by $x \mapsto (x,x)$. ◀

▶ **Lemma 4.10.** *The* bind *map defined by* $\mathrm{bind}(f,\mu) \stackrel{def}{=} f^\dagger(\mu)$ *is non-expansive from* $\mathrm{Dist}\,Y \otimes_1 (Y \to_1 \mathrm{Dist}\,X)$ *to* $\mathrm{Dist}\,X$.

See Appendix A.2 for a proof of the soundness of the typing rule for the primitive operations. Note that the types of the higher-order primitives setmap, setfilter, and setfold are not sound when the functional is not guaranteed to converge. Some solutions to this problem are discussed in [23, Section 3.5].

Recursive types and Recursive functions

In this section, we will show that the introduction of a denotational semantics for interpreting recursive definitions of both data types and functions for the fragment of Fuzz that does not include probability distributions [3] can be generalised to our setting. Note that the interpretation is parametrized by a finite set of type identifiers, that behave as iso-recursive types, and by a definition environment. This approach slightly diverges from what precedes.

▶ **Definition 4.11.** *A* metric complete partial order *is a complete partial order X endowed with a metric d such that for all $(x_i)_{i\in\mathbf{N}}$ and $(x'_i)_{i\in\mathbf{N}}$ two ω-chains in X, if $d_X(x_i, x'_i) \leq r$ for all $i \in \mathbf{N}$, then $d_X\big(\bigsqcup_{i\in\mathbf{N}} x_i, \bigsqcup_{i\in\mathbf{N}} x'_i\big) \leq r$.*

Equivalently, we may ask that $d\big(\bigsqcup_{i\in\mathbf{N}} x_i, \bigsqcup_{i\in\mathbf{N}} {x'}_i\big) \leq \liminf_{i\to\infty} d(x_i, x'_i)$ [3, Lemma 4.5].

This framework allows to describe Plurimetric Fuzz recursive types as solutions to domain equations of the form $F(X) = X$, and to describe divergence by the least element $\bot$.

▶ **Theorem 4.12** ([3, Theorem 4.15])**.** $\mathsf{MetCPO}_\bot$ *is an algebraically compact* CPO*-category, that is for every* CPO*-endofunctor* F*, there exists an object* μF *and an isomorphism* $i : F(\mu F) \simeq \mu F$ *such that* i *is an initial algebra and* i^{-1} *is a final coalgebra.*

We have to show that $\mathsf{MetCPO}_\bot$ is closed under the tensor and arrow constructors.

▶ **Lemma 4.13.** *If* X *and* Y *are two metric complete partial orders, then so is* $X \multimap_p Y$*.*

Proof. Let $(f_i)_{i\in\mathbf{N}}$ and $(g_i)_{i\in\mathbf{N}}$ be two ω-chains in $X \multimap_p Y$ such that for all $i \in \mathbf{N}$, we have $d_{X\multimap_p Y}(f_i, g_i) \leq r$. Let x_1 and x_2 in X, and $i \in \mathbf{N}$.

$$\begin{aligned} d_Y\big(f_i(x_1), g_i(x_2)\big) &\leq d_{X\multimap_1 Y}(f_i, g_i) + d_X(x_1, x_2) && \text{by Equation (4)} \\ &\leq d_{X\multimap_p Y}(f_i, g_i) + d_X(x_1, x_2) && \text{by [27, Theorem 5]} \\ &\leq r + d_X(x_1, x_2) \end{aligned}$$

◀

▶ **Lemma 4.14.** *If* X *and* Y *are two metric complete partial orders, then so is* $X \otimes_p Y$*.*

Proof. Let $(p_i)_{i\in\mathbf{N}}$ and $(p'_i)_{i\in\mathbf{N}}$ be two ω chains in $X \times Y$. For all $i \in \mathbf{N}$ we write $p_i = (x_i, y_i)$ and $p'_i = (x'_i, y'_i)$. Since X and Y are metric complete partial orders, we have

$$d_X\left(\bigsqcup_{i\in\mathbf{N}} x_i, \bigsqcup_{i\in\mathbf{N}} {x'}_i\right) \leq \liminf_{i\to\infty} d_X(x_i, x'_i) \quad \text{and} \quad d_Y\left(\bigsqcup_{i\in\mathbf{N}} y_i, \bigsqcup_{i\in\mathbf{N}} {y'}_i\right) \leq \liminf_{i\to\infty} d_Y(y_i, y'_i)\,.$$

Therefore, as the function $x \mapsto x^p$ is increasing, we have

$$\begin{aligned} d_X\left(\bigsqcup_{i\in\mathbf{N}} x_i, \bigsqcup_{i\in\mathbf{N}} {x'}_i\right)^p + d_Y\left(\bigsqcup_{i\in\mathbf{N}} y_i, \bigsqcup_{i\in\mathbf{N}} {y'}_i\right)^p &\leq \liminf_{i\to\infty} d_X(x_i, x'_i)^p + \liminf_{i\to\infty} d_Y(y_i, y'_i)^p \\ &\leq \liminf_{i\to\infty} \big(d_X(x_i, x'_i)^p + d_Y(y_i, y'_i)^p\big) \\ &= \liminf_{i\to\infty} d_X(x_i, x'_i)^p + d_Y(y_i, y'_i)^p \end{aligned}$$

and by taking the p-th root of both sides, we obtain $d_{X\otimes_p Y}\big(\bigsqcup_{i\in\mathbf{N}} p_i, \bigsqcup_{i\in\mathbf{N}} {p'}_i\big) \leq \liminf_{i\to\infty} d_{X\otimes_p Y}(p_i, p'_i)$. ◀

From Proposition 4.5 and what precedes, we can deduce the soundness of the deterministic fragment of Plurimetric Fuzz, which features recursive types and functions.

▶ **Theorem 4.15** (Soundness)**.** *If* π *is a derivation in the deterministic fragment of Plurimetric Fuzz of* $(p)\ \Gamma \vdash a : A$*, then* $[\![\pi]\!]$ *is a non-expansive map from* $[\![(p)\ \Gamma]\!]$ *to* $[\![A]\!]$*.*

▶ Remark 4.16. One may notice that if we use recursive types to define Nat as $\mu\alpha.\ \mathsf{Unit} \oplus \alpha$, then the following implementation of $(+)$ is non-expansive for all p rather than $c(1,p)$-sensitive:

Listing 1 Non-expansive implementation of the addition.

```
let rec (+) n m = match n with injl () -> m | injr k -> injr (k + m)
```

However, in this setting, the sensitivity is calculated with respect to the following distance: $d_{\mathsf{Nat}}(m, n)$ equals 0 if $m = n$, and ∞ otherwise, rather than the usual distance on $\mathbf{N}$. This justifies the introduction of Nat as a primitive data type and of $(+)$ as a primitive operation. ⌟

5 Metatheoretical properties

Unless otherwise stated, $[\![-]\!]$ will refer to one of the two closely-related denotational semantics we have defined. More specifically, $[\![A]\!]$ may be either a metric space or a metric complete partial order (CPO). Furthermore, the terms will be drawn from the appropriate fragment.

In the same way as [3], we call a *substitution* a finite partial map from variables to values. We write $S(e)$ for the simultaneous substitution of x by $S(x)$ in e for all x in $\mathrm{dom}(S) \cap \mathrm{FV}(e)$. A substitution S is said to be well-typed by a precontext Γ and we write $S : \Gamma$ when the following two assertions are equivalent: $\vdash S(x) : A$ and $[x : A]_s \in \Gamma$ for some $s \geq 0$. Finally, for all parameters p, we naturally define $[\![S]\!]$ as an element of $[\![(p)\ \Gamma]\!]$.

▶ **Lemma 5.1** (Substitution). *For all derivations π of $(p)\ \Gamma, \Delta \vdash a : A$ and for all well-typed substitutions $S : \Gamma$, there exists a derivation π' of $(p)\ \Delta \vdash S(a) : A$. Moreover, $[\![\pi']\!] = [\![\pi]\!]([\![S]\!], -)$.*

Types and denotation are preserved by the operational semantics.

▶ **Theorem 5.2** (Preservation). *For all derivations π_a of $\vdash a : A$, if $a \Downarrow v$, then there exists a derivation π_v of $\vdash v : A$. Moreover, $[\![\pi_a]\!] = [\![\pi_v]\!]$.*

Let us now state the main result of this section, that is the metric preservation theorem.

▶ **Theorem 5.3** (Metric preservation for Core Plurimetric Fuzz). *For all derivations π of $(p)\ \Gamma \vdash a : A$ and for all well-typed substitutions $S, S' : \Gamma$, then there exists well-typed values v and v' such that $S(a) \Downarrow v$ and $S'(a) \Downarrow v'$ and $d_{[\![A]\!]}([\![v]\!], [\![v']\!]) \leq d_{[\![\Gamma]\!]}([\![S]\!], [\![S']\!])$,*

▶ **Theorem 5.4** (Metric preservation for Plurimetric Fuzz). *For all derivations π of $(p)\ \Gamma \vdash a : A$ and for all well-typed substitutions $S, S' : \Gamma$, we have $d_{[\![A]\!]_\perp}([\![S(a)]\!], [\![S'(a)]\!]) \leq d_{[\![\Gamma]\!]}([\![S]\!], [\![S']\!])$.*

By itself, the second formulation of the metric preservation theorem does not constrain the termination behaviour of the two terms $S(a)$ and $S'(a)$. However, the following lemma connects termination from both the operational and denotational perspectives. More details on the implications of this result are given in [3, Section 5].

▶ **Lemma 5.5** (Adequacy for Plurimetric Fuzz). *If $\emptyset \vdash a : A$ and $[\![a]\!] \neq \perp$, then there exists a value v such that $a \Downarrow v$.*

The proofs are similar to the one given in [3], given our soundness results (Proposition 4.5 and Theorem 4.15).

6 Expressive power and Precision

Let us now illustrate the usage of our type system with three examples.

Example: Functions with Multiple Arguments

Let us consider the term $\lambda c.\ \big(\mathbf{let}\ (x, y) = c\ \mathbf{in}\ f(!x, y) + g(x, !y)\big)$ where $f : !_2\mathsf{Real} \otimes_2 \mathsf{Real} \multimap_2 \mathsf{Real}$ and $g : \mathsf{Real} \otimes_2 !_2\mathsf{Real} \multimap_2 \mathsf{Real}$, that is the same example as in [27, Section 5].

$$\dfrac{\begin{array}{cc}\vdots & \vdots \\ (2)\ [x : \mathsf{Real}]_2, [y : \mathsf{Real}]_1 \vdash f(!x, y) : \mathsf{Real} & (2)\ [x : \mathsf{Real}]_1, [y : \mathsf{Real}]_2 \vdash g(x, !y) : \mathsf{Real}\end{array}}{(2)\ \sqrt{2} \cdot ([x : \mathsf{Real}]_{\sqrt{5}}, [y : \mathsf{Real}]_{\sqrt{5}}) \vdash f(!x, y) + g(x, !y) : \mathsf{Real}}\ +$$

This typing derivation shows that, by using Plurimetric Fuzz, we manage to obtain the same sensitivity as with Bunched Fuzz, that is to say $\sqrt{10} \approx 3 + 1/6$, while a naïve extension of Fuzz would overestimate it to 4 [27, Section 5].

Example: Suboptimal sensitivity analysis

We can without difficulty find a term e and a type A such that $\vdash e : A$ in Bunched Fuzz, but $\not\vdash e : A$ in Plurimetric Fuzz. For example, let $e = \lambda x.\ \big((x, x), *\big)$ and $A = !_2 B \multimap_2 (B \otimes_1 B) \otimes_2 \mathsf{Unit}$ for any type B. (Plurimetric Fuzz would require the exponential constructor to be annotated with at least $2\sqrt{2}$.) However, such cases do not seem to appear in practical programs.

Example: Neighbour classification

Let us consider an example using the Euclidean distance L^2. Say that given a database of labelled points in the Euclidean plane, we want to predict the label of a new point x by a majority vote weighted by the distance d to its neighbours (approximately 1 when $d < r$ for a given radius r, and 0 otherwise). Here, we choose the function $\mathsf{weight} : x \mapsto 1 - 1/(1+e^{-4(x-r)})$. This is the complement of a shifted and scaled function (widely used as an activation function in machine learning), which can be soundly added to the language as a primitive of type $\mathsf{Real} \multimap_1 \mathsf{Real}$ (see Lemma 2.6).

A row of the database is represented by the following type: $\mathsf{Row} = \mathsf{Point} \otimes_1 \mathsf{Label}$ where $\mathsf{Point} = \mathsf{Real} \otimes_2 \mathsf{Real}$, and $\mathsf{Label} = \mathsf{Unit} \oplus \cdots \oplus \mathsf{Unit}$. We assume that the coordinates are precise enough so that no two different points have the same coordinates, and that $x = (0, 0)$ (we lose nothing in generality by doing this, since translation is a non-expansive operation on the Euclidean plane).

The algorithm is implemented as follows (where $=$ is an ∞-sensitive primitive):

Listing 2 Implementation of the neighbour classification algorithm.

```
let get_pos   (r : row) : point = let (pos, _)   = r in pos
let get_label (r : row) : label = let (_, label) = r in label

let score (l : label) (db : database) : real = db
        |> setfilter (fun r -> get_label r = l)
        |> setmap (fun r -> distance (0, 0) (get_pos r))
        |> setfold (fun acc x -> acc + weight x) 0

let predict (db : database) : label = exp_noise labels score db
```

Informally, `score` computes the score of a label by: (1) filtering the database to keep only the points with the given label; (2) computing the distance of each point to the origin (the Euclidean distance `distance` is non-expansive on elements of type Point); (3) computing the sum of the weights of the points. Moreover, `exp_noise` is a specialised version of the exponential mechanism presented in [16, Equation 1] for the case $s = 1$ and $\epsilon = 1$, which has type $\mathsf{Set}(\mathsf{Label}) \to_1 (\mathsf{Label} \to_1 \mathsf{Database} \multimap_1 \mathsf{Real}) \to_1 \mathsf{Set}(\mathsf{Row}) \multimap_1 \bigcirc\mathsf{Label}$.

We can derive the following types for the above functions: $\mathtt{score} : \mathsf{Label} \to_1 \mathsf{Database} \multimap_1 \mathsf{Real}$, and $\mathtt{predict} : \mathsf{Set}(\mathsf{Row}) \multimap_1 \bigcirc\mathsf{Label}$. First, by applying the tensor-elimination and arrow-introduction rules, we can show that the helper functions `get_pos` and `get_label` have type $\mathsf{Row} \multimap_1 \mathsf{Point}$ and $\mathsf{Row} \multimap_1 \mathsf{Bool}$ respectively. Then we type the three anonymous functions that appear in our implementation:

- `fun r -> get_label r = l` has type $\mathsf{Row} \to_1 \mathsf{Bool}$ in the context $[l : \mathsf{Label}]_\infty$;
- `fun r -> distance (0, 0) (get_pos r)` has type $\mathsf{Row} \multimap_1 \mathsf{Real}$ as the Euclidean distance `distance` has type $\mathsf{Point} \multimap_1 \mathsf{Point} \multimap_1 \mathsf{Real}$;
- `fun acc x -> acc + weight x` has type $\mathsf{Real} \multimap_1 \mathsf{Real} \multimap_1 \mathsf{Real}$.

This way, we show that `score` has the following type $\mathsf{Label} \to_1 \mathsf{Database} \multimap_1 \mathsf{Real}$ and we can apply the `exp_noise` function to conclude.

In particular, by Lemma 4.3, this classification algorithm is 1-differentially private.

7 Translation Mappings

In order to better understand the relationships between Fuzz and Plurimetric Fuzz we will now investigate some translations between the two systems. We consider a presentation of Fuzz with a weakening rule (W), rather than axioms with an arbitrary context. Moreover, we extend Plurimetric Fuzz by adding a & type constructor to simplify the presentation. Its introduction and elimination rules are given in Appendix B.

Let Der(Fuzz) be the set of derivations in Fuzz endowed with the following partial order: for all derivations π of $\Gamma \vdash a : A$ and π' of $\Delta \vdash b : B$, we have $\pi \leq \pi'$ iff $\Gamma \leq \Delta$ (see Definition 3.4) and $(a, A) = (b, B)$. Similarly, we define Der(PFuzz) for Plurimetric Fuzz.

7.1 Translation from Fuzz to Plurimetric Fuzz

For all parameters p, we define a mapping P^p_{type} from Fuzz types to Plurimetric Fuzz types by structural induction as follows:

$$\begin{aligned} P^p_{\text{type}}(\mathsf{Unit}) &= \mathsf{Unit} & P^p_{\text{type}}(A \multimap B) &= P^p_{\text{type}}(A) \multimap_p P^p_{\text{type}}(B) \\ P^p_{\text{type}}(A \oplus B) &= P^p_{\text{type}}(A) \oplus P^p_{\text{type}}(B) & P^p_{\text{type}}(!_s A) &= !_{s^{1/p}} P^p_{\text{type}}(A) \\ P^p_{\text{type}}(A \,\&\, B) &= P^p_{\text{type}}(A) \,\&\, P^p_{\text{type}}(B) & P^p_{\text{type}}(\bigcirc A) &= \bigcirc P^p_{\text{type}}(A) \\ P^p_{\text{type}}(A \otimes B) &= P^p_{\text{type}}(A) \otimes_p P^p_{\text{type}}(B) & P^p_{\text{type}}(\mu\alpha.\, A) &= \mu\alpha.\, P^p_{\text{type}}(A) \end{aligned} \tag{7}$$

Note that Fuzz lists are mapped to p-lists in Plurimetric Fuzz, i.e., for all type A, we have $P^p_{\text{type}}\big(\mathsf{List}(A)\big) = \mathsf{List}_p\big(P^p_{\text{type}}(A)\big)$. The distance on the latter type is given by $d_{\mathsf{List}_p(A)}(l, l') = \sqrt[p]{\sum_{i=1}^n d_A(l_i, l'_i)^p}$ if $\text{length}(l) = \text{length}(l') = n$, and ∞ otherwise.

We also define a mapping P^p_{ctx} from Fuzz contexts to Plurimetric Fuzz precontexts by $P^p_{\text{ctx}}(\emptyset) = \emptyset$, and $P^p_{\text{ctx}}(\Gamma, [x : A]_s) = P^p_{\text{ctx}}(\Gamma), [x : P^p_{\text{type}}(A)]_{s^{1/p}}$, and a mapping P^p_{der} on derivations. For unary and binary rules, we have for instance:

$$P^p_{\text{der}}\left(\dfrac{}{[x : A]_1 \vdash x : A}\ \text{var}\right) = \dfrac{}{(p)\ [x : P^p_{\text{type}}(A)]_1 \vdash x : P^p_{\text{type}}(A)}\ \text{var}$$

$$P^p_{\text{der}}\left(\dfrac{\begin{matrix}\vdots\,\pi_a \\ \Gamma \vdash a : A\end{matrix} \qquad \begin{matrix}\vdots\,\pi_b \\ \Delta \vdash b : B\end{matrix}}{\Gamma + \Delta \vdash (a, b) : A \otimes B}\ \otimes I\right) = \dfrac{\dfrac{\begin{matrix}\vdots\,P^p_{\text{der}}(\pi_a) \\ (p)\ P^p_{\text{ctx}}(\Gamma) \vdash a : P^p_{\text{type}}(A)\end{matrix} \qquad \begin{matrix}\vdots\,P^p_{\text{der}}(\pi_b) \\ (p)\ P^p_{\text{ctx}}(\Delta) \vdash b : P^p_{\text{type}}(B)\end{matrix}}{(p)\ C^p\left(P^p_{\text{ctx}}(\Gamma); P^p_{\text{ctx}}(\Delta)\right) \vdash (a, b) : P^p_{\text{type}}(A) \otimes_p P^p_{\text{type}}(B)}\ \otimes I}{(p)\ P^p_{\text{ctx}}(\Gamma + \Delta) \vdash (a, b) : P^p_{\text{type}}(A \otimes B)}\ = W$$

▶ **Definition 7.1.** *A derivable judgement $\Gamma \vdash e : A$ is said to be* minimal *in a (Plurimetric) Fuzz if for all contexts Δ such that $\Delta \vdash e : A$, we have $\Gamma \leq \Delta$.*

We can now prove the main result of this section, that is to say that the translation of a valid derivation is valid (see Appendix B for a proof).

▶ **Lemma 7.2.** *For all precontexts Γ and Δ, sensitivities s, and parameters p, we have the following equality: $C^p\left(P^p_{\text{ctx}}(\Gamma); s^{1/p} \cdot P^p_{\text{ctx}}(\Delta)\right) = P^p_{\text{ctx}}(\Gamma + s\Delta)$.*

▶ **Corollary 7.3.** *For all parameters p, the image by P^p_{der} of the derivation π of a (minimal) judgement $\Gamma \vdash a : A$ in Fuzz is a valid derivation of a (minimal) judgement $(p)\ P^p_{\text{ctx}}(\Gamma) \vdash a : P^p_{\text{type}}(A)$ in Plurimetric Fuzz.*

In particular, all examples in [23] that only use structural and logical rules can be translated to Plurimetric Fuzz for any parameter p. This includes elementary operations on lists such as binary and iterated concatenation, length, but also higher-order combinators such as map, foldl, foldr (see [23, Section 3.2]). More generally, this means that the L^1 sensitivity properties obtained by typing in Fuzz for these programs can be for free transposed into L^p sensitivity properties obtained by typing in Plurimetric Fuzz. However, Corollary 7.3 does not extend to primitive operations (the ones presented in Figure 2), which do not behave uniformly with respect to the metric chosen on the pairs and functions.

▷ Claim 7.4 (No miracle). We cannot soundly extend P_{der} to the derivations involving primitive operations such as addition on numbers.

Proof. For instance, we cannot soundly translate the following $(+)$ rule for $p = 2$:

$$\frac{\Gamma \vdash a : \mathsf{Real} \qquad \Delta \vdash b : \mathsf{Real}}{\Gamma + \Delta \vdash a + b : \mathsf{Real}}\,(+) \quad \xrightarrow{P^2_{\mathrm{der}}} \quad \frac{(2)\ P^2_{\mathrm{ctx}}(\Gamma) \vdash a : \mathsf{Real} \qquad (2)\ P^2_{\mathrm{ctx}}(\Delta) \vdash b : \mathsf{Real}}{(2)\ P^2_{\mathrm{ctx}}(\Gamma + \Delta) \vdash a + b : \mathsf{Real}}\,(+)$$

as the following function is not non-expansive: $(+) \colon \mathbf{R} \otimes_2 \mathbf{R} \to \mathbf{R}$ (its sensitivity is $\sqrt{2}$). ◁

7.2 Translation from Plurimetric Fuzz to Fuzz

Conversely, we can define *partial* mappings F^p_{type}, F^p_{ctx} and F^p from Plurimetric Fuzz to Fuzz. We only give the most interesting cases:

$$\begin{aligned}
F^p_{\mathrm{type}}(A \otimes_q B) &= F^p_{\mathrm{type}}(A) \otimes F^p_{\mathrm{type}}(B) && \text{if } q \le p \\
F^p_{\mathrm{type}}(A \multimap_q B) &= F^p_{\mathrm{type}}(A) \multimap F^p_{\mathrm{type}}(B) && \text{if } q \le p \\
F^p_{\mathrm{type}}(!_s A) &= !_{s^p} F^p_{\mathrm{type}}(A) \\
F^p_{\mathrm{ctx}}([x : A]_s, \Gamma) &= [x : F^p_{\mathrm{type}}(A)]_{s^p}, F^p_{\mathrm{ctx}}(\Gamma) \\
F^p_{\mathrm{der}}\Big(\dfrac{}{(q)\ [x : A]_1 \vdash x : A}\ \mathrm{var}\Big) &= \dfrac{}{[x : F^p_{\mathrm{type}}(A)]_1 \vdash x : F^p_{\mathrm{type}}(A)}\ \mathrm{var} && \text{if } q \le p
\end{aligned}$$

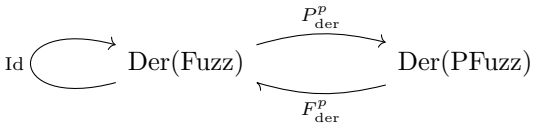
▶ **Lemma 7.5.** *For all precontexts Γ and Δ, for all sensitivities s, we have the following inequality: $s^p F^p_{\mathrm{ctx}}(\Gamma) + F^p_{\mathrm{ctx}}(\Delta) \le F^p_{\mathrm{ctx}}\big(C^p\,(\Gamma; s\Delta)\big)$.*

It follows from the definition above that the image $F^p_{\mathrm{der}}(\pi)$ of a Plurimetric Fuzz derivation π is defined iff any parameter q occurring in a judgement of π is inferior or equal to p.

▶ **Corollary 7.6.** *For all parameters p, if the image by the mapping F^p_{der} of a derivation π in Plurimetric Fuzz is defined, then it is a valid derivation in Fuzz.*

Finally, we obtain the following property relating the two translations:

▶ **Theorem 7.7.** *For all parameters p, we have $F^p_{\mathrm{der}} \circ P^p_{\mathrm{der}} = \mathrm{Id}_{\mathrm{Der(Fuzz)}}$. In other words, the following diagram commutes:*



8 Conclusion and Future Work

We have shown that Plurimetric Fuzz extends the Fuzz language by handling L^p distance, using the types of Bunched Fuzz but with classical typing judgements. This system can be seen as a subsystem of Bunched Fuzz which satisfies type safety. Among its other benefits are the facts that it includes subtyping which relates distances L^p and L^q, and it supports recursive types. We have also investigated translations between Plurimetric Fuzz and Fuzz.

Type checking and type inference for systems based on linear logic have been the object of several works, e.g., [4, 1, 13]. While type checking for Fuzz is straightforward (for DFuzz [16], which is a variant of Fuzz that incorporates dependent types, see [2]), we anticipate that type checking for Plurimetric Fuzz will be significant more challenging to the non-linear nature of the sensitivity constraints. If solved, it would allow us to replace Fuzz by Plurimetric Fuzz in [26], and obtain a type system for adaptive differential privacy with respect to vector metrics.

In addition, one may work on improving the sensitivity obtained by typing in Plurimetric Fuzz. One the one hand, we do not know whether a generalisation of the monad elimination rule to any parameter p, which would be finer than the one presented in this paper, is sound.

Finally, the question of whether one can combine recursive types and functions with probability distributions is still open, both in the case of Fuzz and of its extensions like Plurimetric Fuzz.

References

1 Vincent Atassi, Patrick Baillot, and Kazushige Terui. Verification of Ptime reducibility for system F terms: Type inference in dual light affine logic. *Log. Methods Comput. Sci.*, 3(4), 2007. doi:10.2168/LMCS-3(4:10)2007.

2 Arthur Azevedo de Amorim, Marco Gaboardi, Emilio Jesús Gallego Arias, and Justin Hsu. Really natural linear indexed type checking. In *Proceedings of the 26nd 2014 International Symposium on Implementation and Application of Functional Languages*. Association for Computing Machinery, October 2014. doi:10.1145/2746325.2746335.

3 Arthur Azevedo de Amorim, Marco Gaboardi, Justin Hsu, Shin ya Katsumata, and Ikram Cherigui. A semantic account of metric preservation. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*. Association for Computing Machinery, January 2017. doi:10.1145/3009837.3009890.

4 Patrick Baillot and Martin Hofmann. Type inference in intuitionistic linear logic. In *Proceedings of the 12th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP)*, pages 219–230. ACM, 2010. doi:10.1145/1836089.1836118.

5 Gilles Barthe, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. Proving differential privacy via probabilistic couplings. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 749–758. ACM, 2016. doi:10.1145/2933575.2934554.

6 Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella Béguelin. Probabilistic relational reasoning for differential privacy. In John Field and Michael Hicks, editors, *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012, Philadelphia, Pennsylvania, USA, January 22-28, 2012*, pages 97–110. ACM, 2012. doi:10.1145/2103656.2103670.

7 Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella Béguelin. Probabilistic relational reasoning for differential privacy. *ACM Trans. Program. Lang. Syst.*, 35(3):9:1–9:49, 2013. doi:10.1145/2492061.

8 Gilles Barthe and Federico Olmedo. Beyond differential privacy: Composition theorems and relational logic for f-divergences between probabilistic programs. In *Automata, Languages, and Programming*, pages 49–60. Springer Berlin Heidelberg, 2013. doi:10.1007/978-3-642-39212-2_8.

9 Olivier Bousquet and André Elisseeff. Stability and generalization. *J. Mach. Learn. Res.*, 2:499–526, 2002. URL: http://jmlr.org/papers/v2/bousquet02a.html.

10 Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, March 2004.

11 Clément L. Canonne, Gautam Kamath, and Thomas Steinke. The discrete gaussian for differential privacy. In *Advances in Neural Information Processing Systems*, volume 33, pages 15676–15688. Curran Associates, Inc., 2020.

12 Swarat Chaudhuri, Sumit Gulwani, Roberto Lublinerman, and Sara NavidPour. Proving programs robust. In Tibor Gyimóthy and Andreas Zeller, editors, *SIGSOFT/FSE'11 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-19) and ESEC'11: 13th European Software Engineering Conference (ESEC-13), Szeged, Hungary, September 5-9, 2011*, pages 102–112. ACM, 2011. doi:10.1145/2025113.2025131.

13 Paolo Coppola and Simone Martini. Optimizing optimal reduction: A type inference algorithm for elementary affine logic. *ACM Trans. Comput. Log.*, 7(2):219–260, 2006. doi:10.1145/1131313.1131315.

14 Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith. Calibrating noise to sensitivity in private data analysis. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*, pages 265–284. Springer, 2006. doi:10.1007/11681878_14.

15 Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3–4):211–407, August 2013. doi:10.1561/0400000042.

16 Marco Gaboardi, Andreas Haeberlen, Justin Hsu, Arjun Narayan, and Benjamin C. Pierce. Linear dependent types for differential privacy. *ACM SIGPLAN Notices*, 48(1):357–370, January 2013. doi:10.1145/2480359.2429113.

17 Arpita Ghosh, Tim Roughgarden, and Mukund Sundararajan. Universally utility-maximizing privacy mechanisms. *SIAM Journal on Computing*, 41(6):1673–1693, 2012. doi:10.1137/09076828X.

18 Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–102, 1987.

19 Jean-Yves Girard, Andre Scedrov, and Philip J. Scott. Bounded linear logic: a modular approach to polynomial-time computability. *Theoretical Computer Science*, 97:1–66, 1992. doi:10.1016/0304-3975(92)90386-T.

20 René Gonin and Arthur H. Money. *Nonlinear Lp-Norm Estimation*. Marcel Dekker, Inc., USA, 1989.

21 Joseph P. Near, David Darais, Chike Abuah, Tim Stevens, Pranav Gaddamadugu, Lun Wang, Neel Somani, Mu Zhang, Nikhil Sharma, Alex Shan, and Dawn Song. Duet: an expressive higher-order language and linear type system for statically enforcing differential privacy. *Proceedings of the ACM on Programming Languages*, 3(OOPSLA):1–30, October 2019. doi:10.1145/3360598.

22 Peter W. O'Hearn and David J. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, June 1999. doi:10.2307/421090.

23 Jason Reed and Benjamin C. Pierce. Distance makes the types grow stronger: A calculus for differential privacy. In *Proceedings of the 15th ACM SIGPLAN international conference on Functional programming*. Association for Computing Machinery, September 2010. doi:10.1145/1863543.1863568.

24 Matías Toro, David Darais, Chike Abuah, Joseph P. Near, Damián Árquez, Federico Olmedo, and Éric Tanter. Contextual linear types for differential privacy. *ACM Transactions on Programming Languages and Systems*, 45(2):1–69, May 2023. doi:10.1145/3589207.

25 Larry Wasserman and Shuheng Zhou. A statistical framework for differential privacy. *Journal of the American Statistical Association*, 105(489):375–389, March 2010. doi:10.1198/jasa.2009.tm08651.

26 Daniel Winograd-Cort, Andreas Haeberlen, Aaron Roth, and Benjamin C. Pierce. A framework for adaptive differential privacy. *Proceedings of the ACM on Programming Languages*, 1(ICFP):1–29, August 2017. doi:10.1145/3110254.

27 june wunder, Arthur Azevedo de Amorim, Patrick Baillot, and Marco Gaboardi. Bunched fuzz: Sensitivity for vector metrics. In Thomas Wies, editor, *Programming Languages and Systems: ESOP 2023*, pages 451–478, Cham, April 2023. Springer Nature Switzerland. doi:10.1007/978-3-031-30044-8_17.

Appendix

A Semantics

A.1 Proof of the Soundness of the Typing Rules

▶ **Lemma A.1.** *For all metric spaces X_1, X_2, Y_1, and Y_2, and parameters p, if $f\colon X_1 \to Y_1$ and $g\colon X_2 \to Y_2$ are non-expansive maps, then so is $f \times g\colon X_1 \otimes_p X_2 \to Y_1 \otimes_p Y_2$.*

Proof. Let (x_1, x_2) and (x_1', x_2') be two elements of $X_1 \otimes_p X_2$.

$$\begin{aligned} d_{Y_1\otimes_p Y_2}\big((f\times g)(x_1,x_2),(f\times g)(x_1',x_2')\big) &= d_{Y_1\otimes_p Y_2}\big((f(x_1),g(x_2)),(f(x_1'),g(x_2'))\big) \\ &= \sqrt[p]{d_{Y_1}\big(f(x_1),f(x_1')\big)^p + d_{Y_2}\big(g(x_2),g(x_2')\big)^p} \\ &\le \sqrt[p]{d_{X_1}(x_1,x_1')^p + d_{X_2}(x_2,x_2')^p} \\ &= d_{X_1\otimes_p X_2}\big((x_1,x_2),(x_1',x_2')\big) \end{aligned}$$ ◀

Let us show that bind is non-expansive. We first need the following lemmata.

▶ **Lemma A.2.** *For all finite sequences of positive reals $(x_i)_{1\le i\le n}$ and $(y_i)_{1\le i\le n}$, we have*

$$\frac{\sum_{i=1}^n x_i}{\sum_{i=1}^n y_i} \le \max_{1\le i\le n} \frac{x_i}{y_i} \qquad \textit{and therefore} \qquad \left|\ln \frac{\sum_{i=1}^n x_i}{\sum_{i=1}^n y_i}\right| \le \max_{1\le i\le n} \left|\ln \frac{x_i}{y_i}\right|.$$

Proof. Let us show the first inequality by induction on n.

- If $n = 1$, then the inequality becomes $x_1/y_1 \le \max\{x_1/y_1\}$ which is true.
- If $n = 2$, then we have

$$\begin{aligned} \frac{\sum_{i=1}^n x_i}{\sum_{i=1}^n y_i} &= \frac{x_1}{y_1+y_2} + \frac{x_2}{y_1+y_2} \\ &= \frac{1}{1+\frac{y_2}{y_1}}\cdot\frac{x_1}{y_1} + \frac{1}{1+\frac{y_1}{y_2}}\cdot\frac{x_2}{y_2} = \frac{\frac{1}{y_1}}{\frac{1}{y_1}+\frac{1}{y_2}}\cdot\frac{x_1}{y_1} + \frac{\frac{1}{y_2}}{\frac{1}{y_1}+\frac{1}{y_2}}\cdot\frac{x_2}{y_2}. \end{aligned}$$

Let $u = \frac{\frac{1}{y_1}}{\frac{1}{y_1}+\frac{1}{y_2}}$ and $v = \frac{\frac{1}{y_2}}{\frac{1}{y_1}+\frac{1}{y_2}}$. We have

$$\frac{\sum_{i=1}^n x_i}{\sum_{i=1}^n y_i} \le u\cdot\max\left\{\frac{x_1}{y_1},\frac{x_2}{y_2}\right\} + v\cdot\max\left\{\frac{x_1}{y_1},\frac{x_2}{y_2}\right\} = (u+v)\cdot\max\left\{\frac{x_1}{y_1},\frac{x_2}{y_2}\right\}$$

which is the desired inequality since $u + v = 1$.

- If $n \geq 3$ and if the result has been proved up to $n-1$, then we write

$$\frac{\sum_{i=1}^n x_i}{\sum_{i=1}^n y_i} = \frac{x_1 + \sum_{i=2}^n x_i}{y_1 + \sum_{i=2}^n y_i} \leq \max\left\{\frac{x_1}{y_1}, \frac{\sum_{i=2}^n x_i}{\sum_{i=2}^n y_i}\right\}$$
$$\leq \max\left\{\frac{x_1}{y_1}, \max_{2\leq i\leq n} \frac{x_i}{y_i}\right\} = \max_{1\leq i\leq n} \frac{x_i}{y_i}\,.$$

Now, let us show the second inequality. Let $X = \ln(\sum_{i=1}^n x_i)$ and $Y = \ln(\sum_{i=1}^n y_i)$ so that we have $X - Y = \ln\big(\sum_{i=1}^n x_i\big) - \ln\big(\sum_{i=1}^n y_i\big) = \ln\big(\sum_{i=1}^n x_i / \sum_{i=1}^n y_i\big)$.

- If $X \geq Y$, then $|X-Y| = X - Y$. Moreover, by the first inequality, we have

$$X - Y = \ln \frac{\sum_{i=1}^n x_i}{\sum_{i=1}^n y_i} \leq \ln\left(\max_{1\leq i\leq n} \frac{x_i}{y_i}\right) = \max_{1\leq i\leq n}\left(\ln \frac{x_i}{y_i}\right) \leq \max_{1\leq i\leq n}\left|\ln \frac{x_i}{y_i}\right|.$$

- If $X \leq Y$, then $|X-Y| = Y - X$, and we have

$$Y - X = \ln \frac{\sum_{i=1}^n y_i}{\sum_{i=1}^n x_i} \leq \max_{1\leq i\leq n}\left|\ln \frac{y_i}{x_i}\right| = \max_{1\leq i\leq n}\left|\ln \frac{x_i}{y_i}\right|.$$

In both cases, we get $|X-Y| \leq \max_{1\leq i\leq n} |\ln(x_i/y_i)|$ as desired. ◀

▶ **Lemma A.3.** *The following map is non-expansive:*

$$\begin{array}{rccl} \text{bind} & : & \operatorname{Dist} Y \otimes_1 (Y \to_1 \operatorname{Dist} X) & \longrightarrow & \operatorname{Dist} X \\ & & (\mu, f) & \longmapsto & t \mapsto \sum_{s\in Y} f(s)(t)\,\mu(t)\,. \end{array}$$

We present an elementary proof of this result (which also follows from the work of Barthe and Olmedo [8]).

Proof. Let μ and μ' be two distributions over Y and let f and f' be two maps from Y to $\operatorname{Dist} X$.

For all $t \in X$, we have

$$\left|\ln \frac{\sum_{s\in Y} f(s)(t)\mu(s)}{\sum_{s\in Y} f'(s)(t)\mu'(s)}\right| \leq \max_{s\in Y}\left|\ln \frac{f(s)(t)\mu(s)}{f'(s)(t)\mu'(s)}\right|$$
$$\leq \max_{s\in Y}\left(\left|\ln \frac{\mu(s)}{\mu'(s)}\right| + \left|\ln \frac{f(s)(t)}{f'(s)(t)}\right|\right).$$

Therefore, we have

$$\max_{t\in X}\left|\ln \frac{\sum_{s\in Y} f(s)(t)\mu(s)}{\sum_{s\in Y} f'(s)(t)\mu'(s)}\right| \leq \max_{s\in Y}\left|\ln \frac{\mu(s)}{\mu'(s)}\right| + \max_{s\in Y}\max_{t\in X}\left|\ln \frac{f(s)(t)}{f'(s)(t)}\right|$$

which is equivalent to the desired inequality:

$$\begin{aligned} d_{\operatorname{Dist} X}\big(f^\dagger(\mu), f'^\dagger(\mu')\big) &\leq d_{\operatorname{Dist} Y}(\mu,\mu') + \max_{s\in Y} d_{\operatorname{Dist} X}(f(s), f'(s)) \\ &= d_{\operatorname{Dist} Y}(\mu,\mu') + d_{Y\to_1 \operatorname{Dist} X}(f, f') \\ &= d_{\operatorname{Dist} Y\otimes_1 (Y\to_1 \operatorname{Dist} X)}\big((\mu, f), (\mu', f')\big)\,. \end{aligned}$$

◀

A.2 Proof of the Soundness of the Rules for Primitive Operations

▶ **Lemma A.4.** *The following rules are sound with respect to the denotational semantics:*

$$\frac{(p)\ \Gamma \vdash x : A}{(p)\ \infty \cdot \Gamma \vdash \{x\} : \mathsf{Set}(A)}\ \mathsf{Set} \qquad \frac{(p)\ \Gamma \vdash n : \mathsf{Nat}}{(p)\ 2\Gamma \vdash \{n\} : \mathsf{Set}(\mathsf{Nat})}\ \mathsf{Set}_{\mathsf{Nat}}$$

Proof. For all set X, the sensitivity of the map $x \mapsto \{x\}$ is bounded by ∞. Therefore, on can soundy introduce the following rule:

$$\frac{}{\vdash \lambda x.\ \{x\} : !_\infty A \multimap_1 \mathsf{Set}(A)}$$

which is equiderivable with the (Set) rule.

If $X = \mathbf{N}$, then the map $n \mapsto \{n\}$ is 2-sensitive. Indeed for n and n' in $\mathbf{N}$,

- if $n = n'$, then $\{n\} = \{n'\}$ and therefore $d_{\mathsf{Set}(\mathbf{N})}(\{n\}, \{n'\}) = 0$;
- otherwise, we have $d_{\mathsf{Set}(\mathbf{N})}(\{n\}, \{n'\}) = 2$, and $d_{\mathbf{N}}(n, n') \geq 1$.

In both cases, we get the inequality $d_{\mathsf{Set}(\mathbf{N})}(\{n\}, \{n'\}) \leq 2 \cdot d_{\mathbf{N}}(n, n')$, and for all parameters p, we can soundly introduce the following rule:

$$\frac{}{\vdash \lambda n.\ \{n\} : !_2\mathsf{Nat} \multimap_p \mathsf{Set}(\mathsf{Nat})}$$

which is equiderivable with the ($\mathsf{Set}_{\mathsf{Nat}}$) rule. ◀

▶ Remark. Given a metric space X containing at least one limit point (such as $\mathbf{R}$ with the usual distance), the map $x \mapsto \{x\}$ is ∞-sensitive, and the factor ∞ above is optimal.

B Translation mappings

Below are the typing rules for the & connective that we use in the translation of Fuzz to Plurimetric Fuzz.

$$\frac{(p)\ \Gamma \vdash a : A \quad (p)\ \Gamma \vdash b : B}{(p)\ \Gamma \vdash (a, b) : A\ \&\ B}\ \&I \qquad \frac{(p)\ \Gamma \vdash c : A\ \&\ B}{(p)\ \Gamma \vdash \pi_1(c) : A}\ \&E_\triangleleft \qquad \frac{(p)\ \Gamma \vdash c : A\ \&\ B}{(p)\ \Gamma \vdash \pi_2(c) : B}\ \&E_\triangleright$$

▶ **Lemma B.1.** *For all precontexts Γ, sensitivities s and parameters p, we have:*

- $s^{1/p} \cdot P^p_{\mathrm{ctx}}(\Delta) = P^p_{\mathrm{ctx}}(s\Delta)$;
- $s^p \cdot F^p_{\mathrm{ctx}}(\Delta) = F^p_{\mathrm{ctx}}(s\Delta)$.

Proof. By induction on the structure of Δ. ◀

▶ **Lemma B.2.** *For all precontexts Γ and Δ, for all sensitivities s and parameters p, we have the following equality: $C^p\left(P^p_{\mathrm{ctx}}(\Gamma); s^{1/p} \cdot P^p_{\mathrm{ctx}}(\Delta)\right) = P^p_{\mathrm{ctx}}(\Gamma + s\Delta)$.*

Proof. Let us prove this equality by induction on the structure of Γ.

- If $\Gamma = \emptyset$, then the equality becomes $s^{1/p} \cdot P^p_{\mathrm{ctx}}(\Delta) = P^p_{\mathrm{ctx}}(s\Delta)$.
- If $\Gamma = \Gamma_0, [x : A]_r$, then we write $\Delta = \Delta_0, [x : A]_t$ (with t being possibly zero) and we have

$$\begin{aligned}
C^p\left(P^p_{\mathrm{ctx}}(\Gamma); s^{1/p} \cdot P^p_{\mathrm{ctx}}(\Delta)\right) &= C^p\left(P^p_{\mathrm{ctx}}(\Gamma_0, [x : A]_r); s^{1/p} \cdot P^p_{\mathrm{ctx}}(\Delta_0, [x : A]_t)\right) \\
&= C^p\left(P^p_{\mathrm{ctx}}(\Gamma_0, [x : A]_r); P^p_{\mathrm{ctx}}(s\Delta_0, [x : A]_{st})\right) \\
&= C^p\left(P^p_{\mathrm{ctx}}(\Gamma_0), [x : P^p_{\mathrm{type}}(A)]_{r^{1/p}}; P^p_{\mathrm{ctx}}(s\Delta_0), [x : P^p_{\mathrm{type}}(A)]_{(st)^{1/p}}\right) \\
&= C^p\left(P^p_{\mathrm{ctx}}(\Gamma_0); s^{1/p} \cdot P^p_{\mathrm{ctx}}(\Delta_0)\right), [x : P^p_{\mathrm{type}}(A)]_{\sqrt[p]{r+st}} \\
&= P^p_{\mathrm{ctx}}(\Gamma_0 + s\Delta_0), [x : P^p_{\mathrm{type}}(A)]_{\sqrt[p]{r+st}} \\
&= P^p_{\mathrm{ctx}}(\Gamma_0 + s\Delta_0, [x : A]_{r+st}) \\
&= P^p_{\mathrm{ctx}}(\Gamma + s\Delta)
\end{aligned}$$

as desired. ◀

▶ **Lemma B.3.** *For all precontexts* Γ *and* Δ*, for all sensitivities* s*, for all parameters* p *and* q *such that* $p \geq q$*, we have the following inequality:* $s^p F^p_{\mathrm{ctx}}(\Gamma) + F^p_{\mathrm{ctx}}(\Delta) \leq F^p_{\mathrm{ctx}}\big(C^q\left(\Gamma; s\Delta\right)\big)$.

Proof. Let us prove this inequality by induction on the structure of Γ.

- If $\Gamma = \emptyset$, then the inequality becomes $s^p F^p_{\mathrm{ctx}}(\Delta) \leq F^p_{\mathrm{ctx}}(s\Delta)$.
- If $\Gamma = \Gamma_0, [x : A]_r$, then we write $\Delta = \Delta_0, [x : A]_t$ (with t being possibly zero) and we have

$$\begin{aligned}
s^p F^p_{\mathrm{ctx}}(\Gamma) + F^p_{\mathrm{ctx}}(\Delta) &= s^p F^p_{\mathrm{ctx}}(\Gamma_0, [x : A]_r) + F^p_{\mathrm{ctx}}(\Delta_0, [x : A]_t) \\
&= s^p \big(F^p_{\mathrm{ctx}}(\Gamma_0), [x : F^p_{\mathrm{type}}(A)]_{r^p}\big) + F^p_{\mathrm{ctx}}(\Delta_0), [x : F^p_{\mathrm{type}}(A)]_{t^p} \\
&= s^p F^p_{\mathrm{ctx}}(\Gamma_0) + F^p_{\mathrm{ctx}}(\Delta_0), [x : F^p_{\mathrm{type}}(A)]_{(rs)^p + t^p} \\
&\leq F^p_{\mathrm{ctx}}\big(C^q\left(\Gamma_0; s\Delta_0\right)\big), [x : F^p_{\mathrm{type}}(A)]_{(rs)^p + t^p} \\
&\leq F^p_{\mathrm{ctx}}\big(C^q\left(\Gamma_0; s\Delta_0\right)\big), [x : F^p_{\mathrm{type}}(A)]_{\left((rs)^q + t^q\right)^{p/q}} \\
&\leq F^p_{\mathrm{ctx}}\left(C^q\left(\Gamma_0; s\Delta_0\right), [x : F^p_{\mathrm{type}}(A)]_{\left((rs)^q + t^q\right)^{1/q}}\right) \\
&\leq F^p_{\mathrm{ctx}}\big(C^q\left(\Gamma; s\Delta\right)\big)
\end{aligned}$$

as desired. ◀

Simulating Dependency Pairs by Semantic Labeling

Teppei Saito
JAIST, Nomi, Japan

Nao Hirokawa
JAIST, Nomi, Japan

Abstract

We show that termination proofs by a version of the dependency pair method can be simulated by semantic labeling plus multiset path orders. By incorporating a flattening technique into multiset path orders the simulation result can be extended to the dependency pair method for relative termination, introduced by Iborra et al. This result allows us to improve applicability of their dependency pair method.

2012 ACM Subject Classification Theory of computation → Equational logic and rewriting

Keywords and phrases Term rewriting, Relative termination, Semantic labeling, Dependency pairs

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.13

Supplementary Material *Software (Tool and Experimental Data)*:
`https://www.jaist.ac.jp/project/saigawa/24fscd/`

Funding *Teppei Saito*: JST SPRING Grant Number JPMJSP2102
Nao Hirokawa: JSPS KAKENHI Grant Number JP22K11900

Acknowledgements The second author is grateful to Vincent van Oostrom for his valuable questions and comments on his talk at FSCD 2021. We thank René Thiemann and Jürgen Giesl for telling us about the first correctness proof of the dependency pair method via semantic labeling. Last but certainly not least, Alfons Geser and Aart Middeldorp's help on literature was indispensable for this work.

1 Introduction

Arts and Giesl's *dependency pair method* [4] and Zantema's *semantic labeling* [29] are powerful techniques for analyzing termination of term rewrite systems (TRSs). In this paper we show that the former can be simulated by the latter combined with a restricted version of *multiset path orders* [8, Definition 5] (also known as recursive path orders).

Let us give an informal outlook of the idea by means of examples. The first example is a termination proof by the dependency pair method. *Dependency pairs* are rewrite rules that represent dependencies of recursive function calls in a TRS. Termination of the TRS boils down to the problem of finding a suitable well-founded algebra with interpretations that weakly orient all rules in the TRS and strictly orient all dependency pairs.

▶ **Example 1.** We show the termination of the TRS for division of Peano numbers:

$$x - \mathsf{0} \to x \qquad \mathsf{s}(x) - \mathsf{s}(y) \to x - y \qquad \mathsf{0} \div \mathsf{s}(y) \to \mathsf{0} \qquad \mathsf{s}(x) \div \mathsf{s}(y) \to \mathsf{s}((x-y) \div \mathsf{s}(y))$$

There are three dependency pairs:

$$\mathsf{s}(x) -^\sharp \mathsf{s}(y) \to x -^\sharp y \qquad \mathsf{s}(x) \div^\sharp \mathsf{s}(y) \to x -^\sharp y \qquad \mathsf{s}(x) \div^\sharp \mathsf{s}(y) \to (x-y) \div^\sharp \mathsf{s}(y)$$

Here $-^\sharp$ and $\div^\sharp$ are fresh function symbols. Consider the algebra $\mathcal{A}$ comprising polynomial interpretations over natural numbers:

$$\mathsf{0}_\mathcal{A} = 0 \qquad \mathsf{s}_\mathcal{A}(a) = a + 1 \qquad a -_\mathcal{A} b = a -^\sharp_\mathcal{A} b = a \div_\mathcal{A} b = a \div^\sharp_\mathcal{A} b = a$$

Under the interpretations, all rules in the TRS are weakly oriented, and all dependency pairs are strictly oriented. For instance, orientation of the last dependency pair is verified as follows: $\mathsf{s}_{\mathcal{A}}(a) \div^{\sharp}_{\mathcal{A}} \mathsf{s}_{\mathcal{A}}(b) = a+1 > a = (a -_{\mathcal{A}} b) \div^{\sharp}_{\mathcal{A}} \mathsf{s}_{\mathcal{A}}(b)$. Hence the termination is concluded by the dependency pair method.

Semantic labeling is a transformation method that labels function symbols in rewrite rules with values of their function arguments. Termination of the resulting TRS is equivalent to that of the original TRS.

▶ **Example 2** (continued from Example 1)**.** The termination of the TRS $\mathcal{R}$ can also be shown by semantic labeling. We use the same algebra $\mathcal{A}$ to label $-$ and $\div$ with values of their first arguments. The resulting labeled TRS consists of the rewrite rules

$$\begin{aligned} x -_a \mathsf{0} &\to x & \mathsf{0} \div_0 \mathsf{s}(y) &\to \mathsf{0} \\ \mathsf{s}(x) -_{a+1} \mathsf{s}(y) &\to x -_a y & \mathsf{s}(x) \div_{a+1} \mathsf{s}(y) &\to \mathsf{s}((x -_a y) \div_a \mathsf{s}(y)) \end{aligned}$$

for all $a \in \mathbb{N}$ and the auxiliary rules $x -_a y \to x -_b y$ and $x \div_a y \to x \div_b y$ for all $a, b \in \mathbb{N}$ with $a > b$. For instance, the label $a+1$ in $\mathsf{s}(x) \div_{a+1} \mathsf{s}(y)$ is the value of $\mathsf{s}(x)$ in $\mathcal{A}$ when x is assigned to a. The termination of this TRS is easily verified by the multiset path order with the (quasi-)precedence $\div_{a+1} \approx -_{a+1} \succ \div_a \approx -_a \succ \mathsf{s} \succ \mathsf{0}$ for all $a \in \mathbb{N}$.

In this paper we show that any termination proof by the dependency pair method can be effectively simulated by the combination of semantic labeling and a multiset path order. By incorporating a *flattening* technique (cf. [6]) in multiset path orders, this simulation result can be extended to Iborra et al.'s dependency pair method for relative termination [15]. Exploiting the simulation result, we improve applicability of this method.

An obstacle to the simulation results is a discrepancy between the two formalisms: the basic theorem of the dependency pair method is based on order pairs called *reduction pairs*, while Zantema's semantic labeling is based on well-founded algebras. We overcome this by reformulating semantic labeling and multiset path orders in forms suited for order pairs.

Interestingly, prior to the seminal paper [4], Arts [3] proved a restricted version of the dependency pair method by using Zantema's semantic labeling [29]. This is the first simulation result, and our work can be considered a revisit of the earlier attempt. This time we use Geser's generalized version [12]. Simulations by semantic labeling are not only of theoretical/historical interest but also of practical interest. In fact, based on the simulation result we relax a precondition of the result of Iborra et al. In addition, having proofs of the dependency pair method in a different route might ease formalization in proof assistants or extension to different rewrite formats.

The remaining part of the paper is organized as follows: In Section 2 we recall basic notions for term rewriting and multiset path orders based on order pairs. In order to simulate dependency pairs by Iborra et al. we introduce a variant of semantic labeling for relative termination in Section 3. In Section 4 we show how Arts and Giesl's dependency pair method can be simulated by the combination of semantic labeling and multiset path orders. In Section 5 we do the same for Iborra et al.'s dependency pair method, using a relative termination criterion that originates from multiset path orders and flattening. Correctness of the criterion is proved in Section 6. Exploiting this simulation result, we improve the applicability of the dependency pair method by Iborra et al. in Section 7. Section 8 concludes the paper by discussing experimental results and related work.

2 Preliminaries

Throughout the paper, we assume familiarity with term rewriting [5, 23].

Term Rewriting

Let $\mathcal{F}$ be a signature and $\mathcal{V}$ a countable set of variables with $\mathcal{F} \cap \mathcal{V} = \varnothing$. The set of all *terms* built from $\mathcal{F}$ and $\mathcal{V}$ is referred to as $\mathcal{T}(\mathcal{F}, \mathcal{V})$. When we need to indicate the arity of a function symbol f, we write $f^{(n)}$ for f. A term t is a *function application* if t is of the form $f(t_1, \ldots, t_n)$, and the *root symbol* f is denoted by $\mathsf{root}(t)$. The *size* $|t|$ of a term t is the number of function symbols and variables occurring in t. The set of function symbols or variables occurring in a term t is denoted by $\mathcal{F}\mathsf{un}(t)$ or $\mathcal{V}\mathsf{ar}(t)$, respectively.

Let $\Box$ be a constant with $\Box \notin \mathcal{F}$. *Contexts* are terms over $\mathcal{F} \cup \{\Box\}$ that contain exactly one $\Box$. The term resulting from replacing $\Box$ in a context C by a term t is denoted by $C[t]$. We write $s \trianglerighteq t$ if there is a context C with $s = C[t]$. The strict part of $\trianglerighteq$ is denoted by $\triangleright$. A *substitution* is a mapping σ from variables to terms such that $\{x \in \mathcal{V} \mid \sigma(x) \neq x\}$ is finite. The application $t\sigma$ of a substitution σ to a term t is inductively defined as follows:

$$t\sigma = \begin{cases} \sigma(t) & \text{if } t \text{ is a variable} \\ f(t\sigma_1, \ldots, t\sigma_n) & \text{if } t = f(t_1, \ldots, t_n) \end{cases}$$

We say that a substitution σ is *grounding* for a set T of terms if $t\sigma$ is ground for all $t \in T$. The grounding target T may be omitted when T is clear from the context. A pair (ℓ, r) of terms is said to be a *rewrite rule* if ℓ is not a variable and every variable in r occurs in ℓ. Rewrite rules (ℓ, r) are written as $\ell \to r$. A set of rewrite rules is called a *term rewrite system* (TRS). Let $\mathcal{R}$ be a TRS. The relation $\to_{\mathcal{R}}$ is defined on terms as follows: $s \to_{\mathcal{R}} t$ if there exist a rewrite rule $\ell \to r \in \mathcal{R}$, a context C, and a substitution σ such that $s = C[\ell\sigma]$ and $t = C[r\sigma]$ hold. In particular, when $C = \Box$ we may write $s \xrightarrow{\epsilon}_{\mathcal{R}} t$, which indicates that the rewriting happens at the root position. A term s is called a *normal form* with respect to a relation $\rightsquigarrow$ if there is no term t with $s \rightsquigarrow t$. The set of normal forms is denoted by $\mathsf{NF}(\rightsquigarrow)$. The TRS $\mathcal{R}$ is said to be *terminating* if $\to_{\mathcal{R}}$ is well-founded. *Relative termination* is a generalized notion of termination [11]. Given TRSs $\mathcal{R}$ and $\mathcal{S}$, we write $\to_{\mathcal{R}/\mathcal{S}}$ for the relation $\to_{\mathcal{S}}^* \cdot \to_{\mathcal{R}} \cdot \to_{\mathcal{S}}^*$. If $\to_{\mathcal{R}/\mathcal{S}}$ is well-founded, we say that $\mathcal{R}$ is *(relatively) terminating* with respect to $\mathcal{S}$ (or $\mathcal{R}/\mathcal{S}$ is terminating).

A pair $(\gtrsim, >)$ of a preorder and a strict order on the same set is called an *order pair* if $a > b$ holds whenever $a \gtrsim \cdot > \cdot \gtrsim b$. Here $>$ need not be the strict pair of $\gtrsim$. The order pair is *well-founded* if $>$ is well-founded. Relative termination is often shown by using well-founded order pairs on terms. A relation $\rightsquigarrow$ on terms is *closed under contexts* (or *monotone*) if $C[s] \rightsquigarrow C[t]$ holds whenever $s \rightsquigarrow t$ and C is a context, and it is *closed under substitutions* if $s\sigma \rightsquigarrow t\sigma$ holds whenever $s \rightsquigarrow t$ and σ is a substitution. We say $\rightsquigarrow$ has the *subterm property* if $s \rightsquigarrow t$ whenever $s \triangleright t$. A relation closed under contexts and substitutions is called a *rewrite relation*. A rewrite relation $\gtrsim$ is a *rewrite preorder* if it is a preorder. A rewrite relation $>$ is a *reduction order* if it is a well-founded order. Moreover, the pair $(\gtrsim, >)$ is called a *monotone reduction pair* if in addition they form an order pair. *Reduction pairs* $(\gtrsim, >)$ are akin to monotone reduction pairs, but the only difference is that $>$ may lack monotonicity.

▶ **Proposition 3.** *Let $\mathcal{R}, \mathcal{S}$ be TRSs. Then $\mathcal{R}/\mathcal{S}$ is terminating if and only if there exists a monotone reduction pair $(\gtrsim, >)$ such that $\mathcal{S} \subseteq \gtrsim$ and $\mathcal{R} \subseteq >$.* ◀

Ordered Algebras

Ordered algebras are key ingredients for constructing orders including ones for reduction pairs. An $\mathcal{F}$-*algebra* (or simply an algebra) is a pair $\mathcal{A} = (A, \{f_\mathcal{A}\}_{f\in\mathcal{F}})$, where A is a set called a *carrier*, and $f_\mathcal{A}$ is an n-ary function on A, called the *interpretation function* of a function symbol $f^{(n)} \in \mathcal{F}$. A mapping from $\mathcal{V}$ to A is called an *assignment* for $\mathcal{A}$. The interpretation $[\alpha]_\mathcal{A}(t)$ of a term t under an assignment α is inductively defined as follows:

$$[\alpha]_\mathcal{A}(t) = \begin{cases} \alpha(t) & \text{if } t \text{ is a variable} \\ f_\mathcal{A}([\alpha]_\mathcal{A}(t_1), \dots, [\alpha]_\mathcal{A}(t_n)) & \text{if } t = f(t_1, \dots, t_n) \end{cases}$$

In this paper we are interested in algebras equipped with *order pairs*. Let $\mathcal{A} = (A, \{f_\mathcal{A}\}_{f\in\mathcal{F}})$ be an algebra with A a non-empty set and $(\gtrsim, >)$ an order pair on A. The triple $(\mathcal{A}, \gtrsim, >)$ is called an *ordered algebra*. We say that the ordered algebra is

- *weakly monotone* if $f_\mathcal{A}(a_1, \dots, a_i, \dots, a_n) \gtrsim f_\mathcal{A}(a_1, \dots, b, \dots, a_n)$ for all $f^{(n)} \in \mathcal{F}$, argument positions $1 \leqslant i \leqslant n$, and $a_1, \dots, a_n, b \in A$ with $a_i \gtrsim b$;
- *well-founded* if $>$ is well-founded.

Remark that monotonicity with respect to $>$ is not imposed on the interpretations $f_\mathcal{A}$. We write $s \gtrsim_\mathcal{A} t$ if $[\alpha]_\mathcal{A}(s) \gtrsim [\alpha]_\mathcal{A}(t)$ holds for all assignments α. Similarly, we write $s >_\mathcal{A} t$ if $[\alpha]_\mathcal{A}(s) > [\alpha]_\mathcal{A}(t)$ for all assignments α. The following facts are known:

- $(\gtrsim_\mathcal{A}, >_\mathcal{A})$ is an order pair and both $\gtrsim_\mathcal{A}$ and $>_\mathcal{A}$ are closed under substitutions;
- if $\mathcal{A}$ is weakly monotone then $\gtrsim_\mathcal{A}$ is closed under contexts; and
- if $\mathcal{A}$ is well-founded then $>_\mathcal{A}$ is well-founded.

Therefore, if $\mathcal{A}$ is weakly monotone and well-founded then $(\gtrsim_\mathcal{A}, >_\mathcal{A})$ is a reduction pair.

Multiset Path Orders

We use multiset path orders (MPOs) [8, Definition 5] based on *precedence pairs*, namely order pairs on the signature. The definition employs multiset extensions of order pairs [25] in a recursive way.

Let $(\gtrsim, >)$ be a pair of relations. For multisets X and Y we write $X \gtrsim_\mathsf{mul} Y$ if there are partitions $X = \{x_1, \dots, x_n\} \uplus X'$ and $Y = \{y_1, \dots, y_n\} \uplus Y'$ such that $x_i \gtrsim y_i$ for all $1 \leqslant i \leqslant n$, and for every $y \in Y'$ there exists $x \in X'$ with $x > y$. Furthermore, if in addition $X' \neq \varnothing$, we write $X >_\mathsf{mul} Y$. If $(\gtrsim, >)$ is an order pair, so is $(\gtrsim^\mathsf{mul}, >^\mathsf{mul})$. Moreover, if $>$ is well-founded, so is $>^\mathsf{mul}$.

▶ **Definition 4.** *Let $(\succsim, \succ)$ be a precedence pair on $\mathcal{F}$. The order pair $(\succsim_\mathsf{mpo}, \succ_\mathsf{mpo})$ of the* multiset path orders *is inductively defined on terms over $\mathcal{F}$ as follows:*

- *$s \succ_\mathsf{mpo} t$ if $s = f(s_1, \dots, s_m)$ and one of the following conditions holds.*
  1. *$s_i \succsim_\mathsf{mpo} t$ for some $1 \leqslant i \leqslant m$.*
  2. *$t = g(t_1, \dots, t_n)$, $f \succ g$, and $s \succ_\mathsf{mpo} t_j$ for all $1 \leqslant j \leqslant n$.*
  3. *$t = g(t_1, \dots, t_n)$, $f \succsim g$, and $\{s_1, \dots, s_m\} \succ^\mathsf{mul}_\mathsf{mpo} \{t_1, \dots, t_n\}$.*
- *$s \succsim_\mathsf{mpo} t$ if either s and t are the same variable, or $s = f(s_1, \dots, s_m)$ and one of the following conditions holds.*
  1. *$s_i \succsim_\mathsf{mpo} t$ for some $1 \leqslant i \leqslant m$.*
  2. *$t = g(t_1, \dots, t_n)$, $f \succ g$, and $s \succ_\mathsf{mpo} t_j$ for all $1 \leqslant j \leqslant n$.*
  3. *$t = g(t_1, \dots, t_n)$, $f \succsim g$, and $\{s_1, \dots, s_m\} \succsim^\mathsf{mul}_\mathsf{mpo} \{t_1, \dots, t_n\}$.*

Here $(\succsim^\mathsf{mul}_\mathsf{mpo}, \succ^\mathsf{mul}_\mathsf{mpo})$ stands for the multiset extension of $(\succsim_\mathsf{mpo}, \succ_\mathsf{mpo})$.

A small remark is that the definition above is based on mutual recursion (cf. [26, Definition 4]). Basic properties of MPOs are readily proved.

▶ **Theorem 5.** *For every well-founded precedence pair the induced order pair $(\succsim_\mathsf{mpo}, \succ_\mathsf{mpo})$ is a monotone reduction pair.* ◀

3 Semantic Labeling for Relative Termination

We introduce semantic labeling for *relative termination*. As stated in the introduction, the original version of semantic labeling [29, Theorem 8] (see also [23, Section 6.5.4]) employs a well-founded algebra and labeling functions to reduce termination of a given TRS into termination of the labeled TRS. Using the notion of relative rewriting, Geser [12] got rid of the well-foundedness requirement from employed ordered algebras. Our variant of semantic labeling is a straightforward adaptation of his result to our setting.

Let $\mathcal{A} = (A, \{f_\mathcal{A}\}_{f\in\mathcal{F}})$ be an algebra, $\{L_f\}_{f\in\mathcal{F}}$ a family of non-empty subsets of A, and $\{\mathsf{lab}_f\}_{f\in\mathcal{F}}$ a family of functions where lab_f is a mapping from A^n to L_f for each $f^{(n)} \in \mathcal{F}$. The pair $(\{L_f\}_{f\in\mathcal{F}}, \{\mathsf{lab}_f\}_{f\in\mathcal{F}})$ (denoted by $\mathcal{L}$ in this paper) is called a *labeling* for $\mathcal{A}$. Elements in L_f are called *labels*. For each $f^{(n)} \in \mathcal{F}$ and $a \in L_f$ we introduce a fresh function symbol f_a if $|L_f| > 1$, and if $L_f = \{a\}$ we reuse the original symbol, namely define $f_a = f$. The *labeled signature* $\{f_a^{(n)} \mid f^{(n)} \in \mathcal{F} \text{ and } a \in L_f\}$ is denoted by $\mathcal{F}_\mathsf{lab}$. Note that, in our formulation, symbols f not subject to labeling (i.e., $|L_f| = 1$) are still included in $\mathcal{F}_\mathsf{lab}$. The *labeling function* lab for terms $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ under an assignment $\alpha : \mathcal{V} \to A$ is defined as

$$\mathsf{lab}(t,\alpha) = \begin{cases} t & \text{if } t \text{ is a variable} \\ f_a(\mathsf{lab}(t_1,\alpha),\dots,\mathsf{lab}(t_n,\alpha)) & \text{if } t = f(t_1,\dots,t_n) \end{cases}$$

where in the second case $a = \mathsf{lab}_f([\alpha]_\mathcal{A}(t_1),\dots,[\alpha]_\mathcal{A}(t_n))$. The resulting term $\mathsf{lab}(\alpha, t)$ is a term over $\mathcal{F}_\mathsf{lab}$. Let $\mathcal{R}$ be a TRS over $\mathcal{F}$. The *labeled TRS* $\mathcal{R}_\mathsf{lab}$ is defined as follows:

$$\mathcal{R}_\mathsf{lab} = \{\mathsf{lab}(\ell,\alpha) \to \mathsf{lab}(r,\alpha) \mid \ell \to r \in \mathcal{R} \text{ and } \alpha \text{ is an assignment}\}$$

▶ **Example 6.** Consider the one-rule TRS $\mathcal{R} = \{\mathsf{f}(\mathsf{f}(x)) \to \mathsf{f}(\mathsf{g}(\mathsf{f}(x)))\}$ and also consider the following algebra $\mathcal{A} = (\{0,1\}, \{\mathsf{f}_\mathcal{A}, \mathsf{g}_\mathcal{A}\})$ and labeling $\mathcal{L} = (\{L_\mathsf{f}, L_\mathsf{g}\}, \{\mathsf{lab}_\mathsf{f}, \mathsf{lab}_\mathsf{g}\})$:

$$\mathsf{f}_\mathcal{A}(x) = 1,\ \mathsf{g}_\mathcal{A}(x) = 0 \qquad L_\mathsf{f} = \{0,1\},\ L_\mathsf{g} = \{0\} \qquad \mathsf{lab}_\mathsf{f}(x) = x,\ \mathsf{lab}_\mathsf{g}(x) = x$$

The labeling results in the TRS $\mathcal{R}_\mathsf{lab} = \{\mathsf{f}_1(\mathsf{f}_0(x)) \to \mathsf{f}_0(\mathsf{g}(\mathsf{f}_0(x))),\ \mathsf{f}_1(\mathsf{f}_1(x)) \to \mathsf{f}_0(\mathsf{g}(\mathsf{f}_1(x)))\}$.

Our variant of semantic labeling employs weakly monotone algebras and weakly monotone labelings. Let $(\mathcal{A}, \gtrsim, >)$ be an ordered algebra. We say that a labeling $(\{L_f\}_{f\in\mathcal{F}}, \{\mathsf{lab}_f\}_{f\in\mathcal{F}})$ is *weakly monotone* if $\mathsf{lab}_f(a_1,\dots,a_i,\dots,a_n) \gtrsim \mathsf{lab}_f(a_1,\dots,b,\dots,a_n)$ for all $f^{(n)} \in \mathcal{F}$, argument positions $1 \leqslant i \leqslant n$, and $a_1,\dots,a_n,b \in A$ with $a_i \gtrsim b$. We define the TRS of *decreasing rules* with respect to a binary relation $\rightsquigarrow$ on L as follows:

$$\mathcal{D}\mathsf{ec}(\rightsquigarrow) = \{f_a(x_1,\dots,x_n) \to f_b(x_1,\dots,x_n) \mid f^{(n)} \in \mathcal{F} \text{ and } a,b \in L_f \text{ with } a \rightsquigarrow b\}$$

Here $x_1,\dots,x_n$ are pairwise different variables.

We are ready to state the main theorem of semantic labeling for relative termination. The theorem speaks about weakly monotone algebras $(\mathcal{A}, \gtrsim, >)$ but actually their strict orders $>$ are irrelevant for this theorem. In other words, the theorem holds regardless of how $>$ is like. Therefore, for brevity we may write $(\mathcal{A}, \gtrsim)$ instead of $(\mathcal{A}, \gtrsim, >)$. The proof is found in Appendix A.

▶ **Theorem 7.** *Let $\mathcal{R}$ and $\mathcal{S}$ be TRSs and $(\mathcal{A}, \gtrsim)$ a weakly monotone algebra with $\mathcal{R} \cup \mathcal{S} \subseteq \gtrsim_\mathcal{A}$, and let $\mathcal{L}$ be a weakly monotone labeling for $(\mathcal{A}, \gtrsim)$. Then $\mathcal{R}/\mathcal{S}$ is terminating if and only if $\mathcal{R}_\mathsf{lab}/(\mathcal{S}_\mathsf{lab} \cup \mathcal{D}\mathsf{ec}(\gtrsim))$ is terminating.*

Note that until Section 4 we only use the theorem with $\mathcal{S} = \varnothing$, which coincides with Geser's semantic labeling [12, Corollary 1]. With a small example we illustrate a termination proof based on the theorem.

▶ **Example 8** (continued from Example 6)**.** Let $\gtrsim$ be the quasi-order on $\{0,1\}$ with $1 \gtrsim 0$. Then $(\mathcal{A}, \gtrsim)$ and $\mathcal{L}$ are weakly monotone. Since the inequality $\mathsf{f}_{\mathcal{A}}(\mathsf{f}_{\mathcal{A}}(a)) = 1 \gtrsim 1 = \mathsf{f}_{\mathcal{A}}(\mathsf{g}_{\mathcal{A}}(\mathsf{f}_{\mathcal{A}}(a)))$ holds for all $a \in \{0,1\}$, the inclusion $\mathcal{R} \subseteq \gtrsim_{\mathcal{A}}$ follows. The TRS $\mathcal{D}\mathsf{ec}(\gtrsim)$ consists of the four rules: $\mathcal{D}\mathsf{ec}(\gtrsim) = \{\mathsf{f}_0(x) \to \mathsf{f}_0(x), \mathsf{f}_1(x) \to \mathsf{f}_0(x), \mathsf{f}_1(x) \to \mathsf{f}_1(x), \mathsf{g}(x) \to \mathsf{g}(x)\}$. By taking the MPO with the precedence $\mathsf{f}_1 \succ \mathsf{f}_0 \succ \mathsf{g}$ we obtain the inclusions $\mathcal{R}_{\mathsf{lab}} \subseteq \succ_{\mathsf{mpo}}$ and $\mathcal{D}\mathsf{ec}(\gtrsim) \subseteq \gtrsim_{\mathsf{mpo}}$. Therefore, $\mathcal{R}_{\mathsf{lab}}/\mathcal{D}\mathsf{ec}(\gtrsim)$ is terminating by Theorem 5. Hence, by applying Theorem 7 we conclude termination of $\mathcal{R}$.

The original statement of (quasi-model based) semantic labeling [29, Theorem 8] can be seen as a special case of Theorem 7, which relies on termination rather than relative termination. To see this, recall that a term rewrite system $\mathcal{R} \cup \mathcal{S}$ is terminating if and only if $\mathcal{R}/\mathcal{S}$ and $\mathcal{S}$ are terminating [11].

▶ **Corollary 9** ([29, Theorem 8])**.** *Let $(\mathcal{A}, \geqslant)$ be a weakly monotone well-founded algebra with $\geqslant$ a partial order, $\mathcal{L}$ a weakly monotone labeling for $(\mathcal{A}, \geqslant)$, and $\mathcal{R}$ a TRS with $\mathcal{R} \subseteq \geqslant_{\mathcal{A}}$. Then $\mathcal{R}$ is terminating if and only if $\mathcal{R}_{\mathsf{lab}} \cup \mathcal{D}\mathsf{ec}(>)$ is terminating.*

Proof. By Theorem 7 termination of $\mathcal{R}$ is equivalent to that of $\mathcal{R}_{\mathsf{lab}}/\mathcal{D}\mathsf{ec}(\geqslant)$. Because $\to_{\mathcal{D}\mathsf{ec}(\geqslant)}$ and $\to^{=}_{\mathcal{D}\mathsf{ec}(>)}$ coincide, the latter is equivalent to termination of $\mathcal{R}_{\mathsf{lab}}/\mathcal{D}\mathsf{ec}(>)$. Since $>$ is well-founded, $\mathcal{D}\mathsf{ec}(>)$ is terminating. Therefore, $\mathcal{R}_{\mathsf{lab}}/\mathcal{D}\mathsf{ec}(>)$ is terminating if and only if $\mathcal{R}_{\mathsf{lab}} \cup \mathcal{D}\mathsf{ec}(>)$ is terminating. ◀

4 Simulating Dependency Pairs for Termination

We recall a basic form of Arts and Giesl's dependency pair method. Let $\mathcal{G}$ be a subset of the signature $\mathcal{F}$. Given an n-ary function symbol f in $\mathcal{G}$, we introduce a fresh n-ary function symbol $f^\sharp$ called a *marked symbol*. The set of marked symbols is denoted by $\mathcal{G}^\sharp$. Given a term $t = f(t_1, \dots, t_n)$, we write $t^\sharp$ for the term $f^\sharp(t_1, \dots, t_n)$. For a TRS $\mathcal{R}$ the set $\mathcal{D}_{\mathcal{R}}$ of *defined symbols* are defined by $\mathcal{D}_{\mathcal{R}} = \{f \mid f(t_1, \dots, t_n) \to r \in \mathcal{R}\}$. The difference $\mathcal{F} \setminus \mathcal{D}_{\mathcal{R}}$ is denoted by $\mathcal{C}_{\mathcal{R}}$, and the symbols in $\mathcal{C}_{\mathcal{R}}$ are called *constructor symbols* or just constructors.

▶ **Definition 10.** *Let $\mathcal{R}$ be a TRS over the signature $\mathcal{F}$ and let $\mathcal{G} \subseteq \mathcal{F}$. The TRS $\mathsf{DP}_{\mathcal{G}}(\mathcal{R})$ over $\mathcal{F} \cup \mathcal{G}^\sharp$ is defined by $\mathsf{DP}_{\mathcal{G}}(\mathcal{R}) = \{\ell^\sharp \to t^\sharp \mid \ell \to r \in \mathcal{R},\ r \trianglerighteq t,\ \mathsf{root}(\ell), \mathsf{root}(t) \in \mathcal{G},\ \text{and } \ell \not\trianglerighteq t\}$. The TRS $\mathsf{DP}_{\mathcal{D}_{\mathcal{R}}}(\mathcal{R})$ is abbreviated to $\mathsf{DP}(\mathcal{R})$, and its rules are called* dependency pairs *of $\mathcal{R}$.*

Note that the non-subterm condition $\ell \not\trianglerighteq t$ is due to Dershowitz [9], and it was not included in the original definition of $\mathsf{DP}(\mathcal{R})$ [4].

▶ **Theorem 11** ([4, 13])**.** *A TRS $\mathcal{R}$ is terminating if and only if $\mathcal{R} \subseteq \gtrsim$ and $\mathsf{DP}(\mathcal{R}) \subseteq >$ for some reduction pair $(\gtrsim, >)$.*

▶ Remark 12. Today the termination condition in the theorem is stated as *finiteness* of $(\mathsf{DP}(\mathcal{R}), \mathcal{R})$; see [13] for the definition. Whenever $(\mathcal{P}, \mathcal{R})$ is finite, the relation $\to^*_{\mathcal{R}}$ and the restriction of $\to^+_{\mathcal{P}/\mathcal{R}}$ to $\mathcal{R}$-terminating terms form a reduction pair; the restriction takes care of the so-called *minimality* condition of chains. So finiteness and existence of a suitable reduction pair are equivalent.

For any weakly monotone well-founded algebra $(\mathcal{A}, \gtrsim, >)$ the induced order pair $(\gtrsim_{\mathcal{A}}, >_{\mathcal{A}})$ forms a reduction pair. Conversely, for every reduction pair $(\gtrsim, >)$ the ordered algebra $(\mathcal{A}, \gtrsim, >)$ fulfills the desired properties where $\mathcal{A}$ is the *term algebra* $(\mathcal{T}(\mathcal{F} \cup \mathcal{D}^\sharp_{\mathcal{R}}, \mathcal{V}), \{f_{\mathcal{A}}\}_{f \in \mathcal{F}})$ defined by $f_{\mathcal{A}}(t_1, \dots, t_n) = f(t_1, \dots, t_n)$. Therefore, in the remaining part of the paper we investigate Corollary 13 below instead of Theorem 11.

▶ **Corollary 13.** *A TRS $\mathcal{R}$ is terminating if and only if $\mathcal{R} \subseteq \gtrsim_{\mathcal{A}}$ and $\mathsf{DP}(\mathcal{R}) \subseteq >_{\mathcal{A}}$ for some weakly monotone well-founded $(\mathcal{F} \cup \mathcal{D}^\sharp_{\mathcal{R}})$-algebra $(\mathcal{A}, \gtrsim, >)$.* ◀

▶ **Example 14.** Example 1 is an example of termination proofs by Corollary 13. The algebra $\mathcal{A}$ uses the standard orders on $\mathbb{N}$, and its interpretations are weakly monotone. Therefore $(\geqslant_{\mathcal{A}}, >_{\mathcal{A}})$ is a reduction pair. Since $\mathcal{R} \subseteq \geqslant_{\mathcal{A}}$ and $\mathsf{DP}(\mathcal{R}) \subseteq >_{\mathcal{A}}$ we conclude the termination by Corollary 13.

We exemplify how proofs based on the dependency pair method can be simulated by semantic labeling with MPOs, introducing a few necessary definitions.

▶ **Definition 15.** *Let $\mathcal{R}$ be a TRS over the signature $\mathcal{F}$ and $(\mathcal{A}, \gtrsim, >)$ an ordered $(\mathcal{F} \cup \mathcal{D}^\sharp_{\mathcal{R}})$-algebra on a carrier A. Fix an arbitrary element $\bullet \in A$. We define the* $\mathcal{A}$-induced labeling $\mathcal{L}_{\mathcal{A}} = (\{L_f\}_{f \in \mathcal{F}}, \{\mathsf{lab}_f\}_{f \in \mathcal{F}})$ *as follows:*

$$L_f = \begin{cases} A & \text{if } f \in \mathcal{D}_{\mathcal{R}} \\ \{\bullet\} & \text{otherwise} \end{cases} \qquad \mathsf{lab}_f(a_1, \dots, a_n) = \begin{cases} f^\sharp_{\mathcal{A}}(a_1, \dots, a_n) & \text{if } f \in \mathcal{D}_{\mathcal{R}} \\ \bullet & \text{otherwise} \end{cases}$$

We also define the precedence pair $(\succsim, \succ)$, called $\mathcal{A}$-induced precedence pair*, on the labeled signature as follows:*

- *$f_a \succsim g_b$ if either $f, g \in \mathcal{D}_{\mathcal{R}}$ and $a \gtrsim b$, or $g \in \mathcal{C}_{\mathcal{R}}$*
- *$f_a \succ g_b$ if either $f, g \in \mathcal{D}_{\mathcal{R}}$ and $a > b$, or $f \in \mathcal{D}_{\mathcal{R}}$ and $g \in \mathcal{C}_{\mathcal{R}}$*

So in the precedence pair, constructors are smaller than (labeled) defined symbols.

▶ **Example 16** (continued from Example 14, see also Example 2)**.** The $\mathcal{A}$-induced labeling gives the labeled TRS $\mathcal{R}_{\mathsf{lab}}$ consisting of the rules

$$x -_a \mathsf{0} \to x \qquad \mathsf{0} \div_0 \mathsf{s}(y) \to \mathsf{0}$$
$$\mathsf{s}(x) -_{a+1} \mathsf{s}(y) \to x -_a y \qquad \mathsf{s}(x) \div_{a+1} \mathsf{s}(y) \to \mathsf{s}((x - y) \div_a \mathsf{s}(y))$$

for all $a \in \mathbb{N}$. The TRS $\mathcal{D}\mathsf{ec}(\geqslant)$ is also the infinite set consisting of the rules

$$x -_a y \to x -_b y \qquad x \div_a y \to x \div_b y$$

for all $a, b \in \mathbb{N}$ with $a \geqslant b$. By Theorem 7 the termination of $\mathcal{R}$ follows if we show that of $\mathcal{R}_{\mathsf{lab}}/\mathcal{D}\mathsf{ec}(\geqslant)$. The $\mathcal{A}$-induced precedence pair $(\succsim, \succ)$ satisfies $\div_{a+1} \approx -_{a+1} \succ \div_a \approx -_a \succ \mathsf{s} \succ \mathsf{0}$ for all $a \in \mathbb{N}$. Here $f \approx g$ stands for $f \succsim g$ and $g \succsim f$. It is easy to see $\mathcal{R}_{\mathsf{lab}} \subseteq \succ_{\mathsf{mpo}}$ and $\mathcal{D}\mathsf{ec}(\geqslant) \subseteq \succsim_{\mathsf{mpo}}$. Hence, $\mathcal{R}$ is terminating by Theorem 5.

Although a multiset path order is used in the last example, other path orders such as lexicographic path orders (LPOs) [17] can also be used for showing termination of $\mathcal{R}_{\mathsf{lab}}/\mathcal{D}\mathsf{ec}(\gtrsim)$. In order to manifest this fact, we introduce a minimalistic termination criterion, inspired by precedence termination (cf. [20, Lemma 1]).

▶ **Definition 17.** *Let $(\succsim, \succ)$ be a precedence pair and $\mathcal{G} \subseteq \mathcal{F}$. The relation $\succ_{\mathcal{G}}$ on terms is inductively defined as follows: $s \succ_{\mathcal{G}} t$ if $s = f(s_1, \dots, s_m)$, $f \in \mathcal{F} \setminus \mathcal{G}$, and one of the following two conditions holds.*

(1) $s \rhd t$.
(2) $t = g(t_1, \dots, t_n)$, $f \succ g$, *and* $s \succ_{\mathcal{G}} t_j$ *for all* $1 \leqslant j \leqslant n$.
The relation $\succsim_{\mathcal{G}}$ *on terms is defined as follows:* $s \succsim_{\mathcal{G}} t$ *if* $s = f(t_1, \dots, t_n)$, $t = g(t_1, \dots, t_n)$, $f \in \mathcal{F} \setminus \mathcal{G}$, *and* $f \succsim g$.

Due to the minimalistic definition, rules like commutativity $f(x, y) \to f(y, x)$ cannot be ordered by $\succsim_{\mathcal{G}}$.

▶ **Lemma 18.** *If* $s \succ_{\mathcal{G}} t$ *or* $s \succsim_{\mathcal{G}} t$ *then* $s \succ_{\mathsf{mpo}} t$ *or* $s \succsim_{\mathsf{mpo}} t$, *respectively.* ◀

In general, $\succ_{\mathcal{G}}$ and $\succsim_{\mathcal{G}}$ do not form a monotone reduction pair. However, they give the following simple criterion for relative termination.

▶ **Proposition 19.** *Let* $(\succsim, \succ)$ *be a well-founded precedence pair. Then* $\mathcal{R}/\mathcal{S}$ *is terminating if there exists a subset* $\mathcal{G}$ *of* $\mathcal{F}$ *such that* $\mathcal{R} \subseteq \succ_{\mathcal{G}}$ *and* $\mathcal{S} \subseteq \succsim_{\mathcal{G}}$.

Proof. By Lemma 18 and Theorem 5. ◀

We establish the main result of this section, using Proposition 19 with $\mathcal{G} = \mathcal{C}_{\mathcal{R}}$. Note that $\mathcal{C}_{\mathcal{R}}$ is included in $\mathcal{F}_{\mathsf{lab}}$ because $\mathcal{L}_{\mathcal{A}}$ do not label constructor symbols of $\mathcal{R}$.

▶ **Lemma 20.** *Suppose* $\mathsf{DP}(\mathcal{R}) \subseteq >_{\mathcal{A}}$ *and* $\mathcal{G} = \mathcal{C}_{\mathcal{R}}$, *and consider the* $\mathcal{A}$*-induced labeling and the* $\mathcal{A}$*-induced precedence* $(\succsim, \succ)$. *If* $\ell \to r \in \mathcal{R}$ *then* $\mathsf{lab}(\ell, \alpha) \succ_{\mathcal{G}} \mathsf{lab}(t, \alpha)$ *for all subterms* t *of* r *and assignments* α.

Proof. Suppose $\ell \to r \in \mathcal{R}$ and $r \trianglerighteq t$. Let α be an assignment. We show $\mathsf{lab}(\ell, \alpha) \succ_{\mathcal{G}} \mathsf{lab}(t, \alpha)$ by structural induction on t. Because $\ell \to r$ is a rewrite rule, ℓ must be of form $f(\ell_1, \dots, \ell_m)$ with $f \in \mathcal{D}_{\mathcal{R}}$. If $\ell \rhd t$ then $\mathsf{lab}(\ell, \alpha) \rhd \mathsf{lab}(t, \alpha)$ and thereby $\mathsf{lab}(\ell, \alpha) \succ_{\mathcal{G}} \mathsf{lab}(t, \alpha)$. Otherwise, t is not a variable because $\ell \rhd t$ follows from $t \in \mathcal{V}\mathsf{ar}(r) \subseteq \mathcal{V}\mathsf{ar}(\ell)$ and $\ell \notin \mathcal{V}$. So suppose $t = g(t_1, \dots, t_n)$. By the induction hypothesis $\mathsf{lab}(\ell, \alpha) \succ_{\mathcal{G}} \mathsf{lab}(t_j, \alpha)$ for all $j \in \{1, \dots, n\}$. We have $\mathsf{lab}(\ell, \alpha) = f_a(\mathsf{lab}(\ell_1, \alpha), \dots, \mathsf{lab}(\ell_m, \alpha))$ where $a = f^\sharp_{\mathcal{A}}([\alpha]_{\mathcal{A}}(\ell_1), \dots, [\alpha]_{\mathcal{A}}(\ell_m))$. We distinguish two cases, depending on g.

- If $g \notin \mathcal{D}_{\mathcal{R}}$ then $f_a \succ g$. Therefore $\mathsf{lab}(\ell, \alpha) \succ_{\mathcal{G}} g(\mathsf{lab}(t_1, \alpha), \dots, \mathsf{lab}(t_n, \alpha)) = \mathsf{lab}(t, \alpha)$.
- If $g \in \mathcal{D}_{\mathcal{R}}$ then $\ell^\sharp \to t^\sharp \in \mathsf{DP}(\mathcal{R})$ because of $\ell \not\rhd t$. From $\mathsf{DP}(\mathcal{R}) \subseteq >_{\mathcal{A}}$ we obtain $\ell^\sharp >_{\mathcal{A}} t^\sharp$. So by the definition of $>_{\mathcal{A}}$ we obtain $a > b$ for $b = g^\sharp_{\mathcal{A}}([\alpha]_{\mathcal{A}}(t_1), \dots, [\alpha]_{\mathcal{A}}(t_n))$, and thus $f_a \succ g_b$ follows. Therefore $\mathsf{lab}(\ell, \alpha) \succ_{\mathcal{G}} g_b(\mathsf{lab}(t_1, \alpha), \dots, \mathsf{lab}(t_n, \alpha)) = \mathsf{lab}(t, \alpha)$. ◀

▶ **Theorem 21.** *Let* $\mathcal{R}$ *be a TRS and* $(\mathcal{A}, \gtrsim, >)$ *a weakly monotone well-founded algebra with* $\mathcal{R} \subseteq \gtrsim_{\mathcal{A}}$. *The following statements hold for* $\mathcal{G} = \mathcal{C}_{\mathcal{R}}$.
1. *The* $\mathcal{A}$*-induced labeling is a weakly monotone labeling for* $(\mathcal{A}, \gtrsim)$.
2. *The* $\mathcal{A}$*-induced precedence pair* $(\succsim, \succ)$ *is well-founded.*
3. *If* $\mathsf{DP}(\mathcal{R}) \subseteq >_{\mathcal{A}}$ *then* $\mathcal{R}_{\mathsf{lab}} \subseteq \succ_{\mathcal{G}}$.
4. $\mathcal{D}\mathsf{ec}(\gtrsim) \subseteq \succsim_{\mathcal{G}}$.

Proof. The third claim follows from Lemma 20. The other claims are straightforward. ◀

Theorem 21 states that, given any termination proof by the basic dependency pair method (Corollary 13), one can construct a corresponding termination proof by semantic labelling (Theorem 7) and precedence-based termination (Proposition 19). This simulation result is conceivable as an alternative proof for Theorem 11. While the standard correctness proof of the dependency pair method relies on the notion of minimal non-terminating term [4], the one via semantic labeling directly captures the decreasing measure (i.e., labels of defined symbols) by recursive path orders such as MPOs and LPOs.

We conclude the section by stating why we adopted Geser's version of semantic labeling. Since the original semantic labeling (Corollary 9) employs a weakly monotone well-founded algebra, from a given reduction pair $(\gtrsim, >)$ we need to construct a single well-founded partial order that plays *both* roles of $\gtrsim$ and $>$. Geser's version resolves this discrepancy, hiding $>$ behind the relative termination condition of labeled systems.

5 Simulating Dependency Pairs for Relative Termination

Iborra et al. [15] developed a natural extension of the dependency pair method to relative termination. We show that this extension can also be simulated by semantic labeling. First we recall their main theorem.

▶ **Definition 22.** *Let $\mathcal{R}$ and $\mathcal{S}$ be TRSs. We say that $\mathcal{R}$* dominates *$\mathcal{S}$ if $\mathcal{F}\mathsf{un}(r) \cap \mathcal{D}_{\mathcal{R}} = \varnothing$ for all $\ell \to r \in \mathcal{S}$. Let $|t|_x$ denote the number of occurrences of a variable x in a term t. A pair (ℓ, r) of terms is called* non-duplicating *if $|\ell|_x \geqslant |r|_x$ for all variables x, and a rule $\ell \to r$ is* non-duplicating *if (ℓ, r) is so. Finally, a TRS $\mathcal{R}$ is* non-duplicating *if every rule in $\mathcal{R}$ is non-duplicating.*

▶ **Theorem 23** ([15, Theorem 2])**.** *Suppose that a TRS $\mathcal{R}$ dominates a non-duplicating TRS $\mathcal{S}$. Then $\mathcal{R}/\mathcal{S}$ is terminating if and only if $\mathsf{DP}(\mathcal{R}) \subseteq >_{\mathcal{A}}$ and $\mathcal{R} \cup \mathcal{S} \subseteq \gtrsim_{\mathcal{A}}$ for some weakly monotone well-founded algebra $(\mathcal{A}, \gtrsim, >)$.*

Theorem 23 is a generalization of the basic dependency pair method (Corollary 13), since the empty TRS is non-duplicating and dominated by any TRS.

▶ **Example 24.** Recall the TRS $\mathcal{R}$ of division from Example 1. We show the relative termination of $\mathcal{R}$ with respect to the TRS $\mathcal{S} = \{\mathsf{rand}(x) \to x, \mathsf{rand}(x) \to \mathsf{rand}(\mathsf{s}(x))\}$. Since the TRS $\mathcal{S}$ is non-duplicating and $\mathcal{R}$ dominates $\mathcal{S}$, we may use Theorem 23 to show termination of $\mathcal{R}/\mathcal{S}$. The set $\mathsf{DP}(\mathcal{R})$ consists of the three rules, see Example 1. Let $(\mathcal{A}, \geqslant, >)$ be the weakly monotone well-founded algebra, where the carrier consists of ordinal numbers below ω^2 and the interpretations are given by the equations:

$$0_{\mathcal{A}} = 0 \quad \mathsf{s}_{\mathcal{A}}(a) = a + 1 \quad \mathsf{rand}_{\mathcal{A}}(a) = a + \omega \quad a -_{\mathcal{A}} b = a -^{\sharp}_{\mathcal{A}} b = a \div_{\mathcal{A}} b = a \div^{\sharp} b = a$$

It is easy to verify $\mathcal{R} \cup \mathcal{S} \subseteq \geqslant_{\mathcal{A}}$ and $\mathsf{DP}(\mathcal{R}) \subseteq >_{\mathcal{A}}$. For instance, the last rules in $\mathsf{DP}(\mathcal{R})$ and $\mathcal{S}$ are oriented as the inequalities

$$\begin{aligned} \mathsf{s}_{\mathcal{A}}(a) \div^{\sharp}_{\mathcal{A}} \mathsf{s}_{\mathcal{A}}(b) &= a + 1 > a = (a -_{\mathcal{A}} b) \div^{\sharp}_{\mathcal{A}} \mathsf{s}_{\mathcal{A}}(b) \\ \mathsf{rand}_{\mathcal{A}}(a) &= a + \omega = a + 1 + \omega = \mathsf{rand}_{\mathcal{A}}(\mathsf{s}_{\mathcal{A}}(a)) \end{aligned}$$

hold for all ordinals $a, b < \omega^2$. Hence, $\mathcal{R}$ is terminating.

For showing an analog of Theorem 21 in a relative termination setting, from a given reduction pair $(\geqslant_{\mathcal{A}}, >_{\mathcal{A}})$ we construct the $\mathcal{A}$-induced labeling and precedence in the same way. However, existing syntactical termination methods, such as precedence-based termination (Proposition 19) and MPOs, are still incapable of showing termination of labeled systems due to problematic rules in relative systems like $\mathsf{rand}(x) \to \mathsf{rand}(\mathsf{s}(x))$.

▶ **Example 25** (continued from Example 24)**.** Following the construction of Theorem 21, we obtain the TRS $\mathcal{R}_{\mathsf{lab}}$ consisting of

$$\begin{aligned} x -_a 0 &\to x & 0 \div_0 \mathsf{s}(y) &\to 0 \\ \mathsf{s}(x) -_{a+1} \mathsf{s}(y) &\to x -_a y & \mathsf{s}(x) \div_{a+1} \mathsf{s}(y) &\to \mathsf{s}((x - y) \div_a \mathsf{s}(y)) \end{aligned}$$

for all ordinals $a < \omega^2$. The TRS $\mathcal{S}_{\mathsf{lab}}$ is the same as $\mathcal{S}$ since we do not label constructors, and the TRS $\mathcal{D}\mathsf{ec}(\geqslant)$ consists of

$$x -_a y \to x -_b y \qquad\qquad x \div_a y \to x \div_b y$$

for all ordinals $a, b < \omega^2$ with $a \geqslant b$. The inclusions $\mathcal{R}_{\mathsf{lab}} \subseteq \succ_{\mathsf{mpo}}$ and $\mathcal{D}\mathsf{ec}(\geqslant) \subseteq \succsim_{\mathsf{mpo}}$ hold, but $\mathcal{S}_{\mathsf{lab}} \subseteq \succsim_{\mathsf{mpo}}$ does not. Actually, any monotone reduction pair $(\gtrsim, >)$ satisfying the subterm property $\rhd \subseteq >$ is unable to orient $\mathcal{S}_{\mathsf{lab}}$.

We overcome this problem by *flattening* inspired by [6] and [15, Definition 3]. For each $k \in \mathbb{N}$ we introduce a fresh k-ary function symbol c_k, called *compound symbols*. The set of all compound symbols is referred to as $\mathcal{F}_{\mathsf{c}}$.

▶ **Definition 26.** *Let $\mathcal{G}$ be a set of function symbols in $\mathcal{F}$. The* flattening TRS $\mathsf{F}(\mathcal{G})$ *over the signature $\mathcal{F} \cup \mathcal{F}_{\mathsf{c}}$ consists of the rules $f(x_1, \dots, x_n) \to \mathsf{c}_n(x_1, \dots, x_n)$, $\mathsf{c}_1(x) \to x$ and*

$$\mathsf{c}_{k+n+1}(x_1, \dots, x_k, \mathsf{c}_m(y_1, \dots, y_m), z_1, \dots, z_n) \to \mathsf{c}_{k+m+n}(x_1, \dots, x_k, y_1, \dots, y_m, z_1, \dots, z_n)$$

for all $k, m, n \in \mathbb{N}$ and $f^{(n)} \in \mathcal{G}$. Since the TRS is terminating and confluent, every term t admits exactly one normal form, which we denote by $t{\downarrow_{\mathcal{G}}}$. Such a normal form is called a flattened term. *We abbreviate $t{\downarrow_{\mathcal{G}}}$ to $t{\downarrow}$ whenever $\mathcal{G}$ is clear from the context.*

▶ **Example 27** (continued from Example 25)**.** Let $\mathcal{G} = \mathcal{C}_{\mathcal{R}} = \{\mathsf{0}, \mathsf{s}, \mathsf{rand}\}$. For example, the term $t = \mathsf{s}(\mathsf{rand}(\mathsf{0})) -_{\omega+1} \mathsf{s}(\mathsf{0})$ admits the following rewrite sequence of $\mathsf{F}(\mathcal{G})$:

$$\mathsf{s}(\mathsf{rand}(\mathsf{0})) -_{\omega+1} \mathsf{s}(\mathsf{0}) \;\to\; \mathsf{c}_1(\mathsf{rand}(\mathsf{0})) -_{\omega+1} \mathsf{s}(\mathsf{0}) \;\to\; \mathsf{rand}(\mathsf{0}) -_{\omega+1} \mathsf{s}(\mathsf{0}) \;\to^*\; \mathsf{c}_0 -_{\omega+1} \mathsf{c}_0$$

Thus, we obtain $t{\downarrow} = \mathsf{c}_0 -_{\omega+1} \mathsf{c}_0$.

▶ **Example 28.** To see what happens if binary symbols and constant symbols are flattened, let us consider the signature $\mathcal{F} = \{\mathsf{a}^{(0)}, \mathsf{b}^{(0)}, \mathsf{f}^{(2)}\}$ and its subset $\mathcal{G} = \{\mathsf{a}, \mathsf{f}\}$ of $\mathcal{F}$. The terms $\mathsf{f}(\mathsf{f}(\mathsf{a}, x), \mathsf{a})$, $\mathsf{f}(\mathsf{f}(\mathsf{a}, x), \mathsf{b})$, $\mathsf{f}(\mathsf{f}(\mathsf{b}, x), \mathsf{b})$ are flattened into x, $\mathsf{c}_2(x, \mathsf{b})$, $\mathsf{c}_3(\mathsf{b}, x, \mathsf{b})$, respectively.

Since flattening introduces compound symbols in $\mathcal{F}_{\mathsf{c}}$, we extend the $\mathcal{A}$-induced precedence pair $(\succsim, \succ)$ on $\mathcal{F}_{\mathsf{lab}}$ by adjoining all compound symbols as minimal elements. To be precise, the extended precedence pair $(\succsim', >')$ is given by the following conditions:

- $f \succsim' g$ if $f \succsim g$ or $g \in \mathcal{F}_{\mathsf{c}}$.
- $f \succ' g$ if $f \succ g$, or $f \in \mathcal{F}_{\mathsf{lab}}$ and $g \in \mathcal{F}_{\mathsf{c}}$.

Obviously, the pair $(\succsim', \succ')$ is a precedence pair satisfying $\succsim \subseteq \succsim'$ and $\succ \subseteq \succ'$, and well-founded if $(\succsim, \succ)$ is so.

The key observation is that, any rewrite sequence of $\mathcal{R}_{\mathsf{lab}}/\mathcal{S}_{\mathsf{lab}}$ gives rise to a corresponding rewrite sequence of $(\succsim'_{\mathsf{mpo}}, \succ'_{\mathsf{mpo}})$.

▶ **Example 29** (again continued from Example 25)**.** Consider the rewrite sequence:

$$\mathsf{s}(\mathsf{rand}(\mathsf{0})) -_{\omega+1} \mathsf{s}(\mathsf{0}) \;\to_{\mathcal{R}_{\mathsf{lab}}}\; \mathsf{rand}(\mathsf{0}) -_{\omega} \mathsf{0} \;\to_{\mathcal{S}_{\mathsf{lab}}}\; \mathsf{rand}(\mathsf{s}(\mathsf{0})) -_{\omega} \mathsf{0}$$

Let $\mathcal{G} = \mathcal{C}_{\mathcal{R}} = \{\mathsf{rand}, \mathsf{s}, \mathsf{0}\}$. Flattening turns the rewrite sequence into the descending sequence of MPO, namely $\mathsf{c}_0 -_{\omega+1} \mathsf{c}_0 \succ'_{\mathsf{mpo}} \mathsf{c}_0 -_{\omega} \mathsf{c}_0 \succsim'_{\mathsf{mpo}} \mathsf{c}_0 -_{\omega} \mathsf{c}_0$.

For formally discussing the correspondence above, we introduce a relative termination criterion akin to Proposition 19 in Section 4. This criterion exploits the fact that $s \succ_{\mathcal{G}} t$ implies $s{\downarrow_{\mathcal{G}}} \succ'_{\mathsf{mpo}} t{\downarrow_{\mathcal{G}}}$ when $\mathcal{R} \subseteq \succ_{\mathcal{G}}$. The proof is discussed in the next section.

▶ **Theorem 30.** *Let $\mathcal{R}$ and $\mathcal{S}$ be TRSs over a signature $\mathcal{F}$. Then $\mathcal{R}/\mathcal{S}$ is terminating if there exist a well-founded precedence pair $(\succ, \succsim)$ and a subset $\mathcal{G}$ of $\mathcal{F}$ such that $\mathcal{R} \subseteq \succ_{\mathcal{G}}$, $\mathcal{S}$ is non-duplicating, and we have $\ell \succsim_{\mathcal{G}} r$ or $r \in \mathcal{T}(\mathcal{G}, \mathcal{V})$ for all $\ell \to r \in \mathcal{S}$.*

We arrive at the simulation result for the relative version of the dependency pair method.

▶ **Theorem 31.** *Let $\mathcal{R}$ be a TRS and $(\mathcal{A}, \gtrsim, >)$ a weakly monotone well-founded algebra with $\mathcal{R} \cup \mathcal{S} \subseteq \gtrsim_{\mathcal{A}}$. The following statements hold for $\mathcal{G} = \mathcal{C}_{\mathcal{R}}$.*

1. *The $\mathcal{A}$-induced labeling is a weakly monotone labeling for $(\mathcal{A}, \gtrsim)$.*
2. *The $\mathcal{A}$-induced precedence pair $(\succsim, \succ)$ is well-founded.*
3. *If $\mathsf{DP}(\mathcal{R}) \subseteq >_{\mathcal{A}}$ then $\mathcal{R}_{\mathsf{lab}} \subseteq \succ_{\mathcal{G}}$.*
4. *If $\mathcal{S}$ is non-duplicating then so is $\mathcal{S}_{\mathsf{lab}}$.*
5. *If $\mathcal{R}$ dominates $\mathcal{S}$ then $\mathcal{R}_{\mathsf{lab}}$ also dominates $\mathcal{S}_{\mathsf{lab}}$.*
6. *$\mathcal{D}\mathsf{ec}(\gtrsim) \subseteq \succsim_{\mathcal{G}}$.*

Proof. Analogous to the proof of Theorem 21. ◀

Theorem 31 is indeed an analog of Theorem 21. Suppose that relative termination is shown by the dependency pair method with a reduction pair (Theorem 23). Relative termination of the labeled systems resulting from semantic labeling is shown by the precedence-based termination criterion (Theorem 30). The employed labeling and precedence pair are constructible from the reduction pair (Definition 15).

6 Multiset Path Orders with Flattening

This section is devoted to proving Theorem 30, which is obtained as a corollary of two key theorems. Let $(\succsim, \succ)$ be a well-founded precedence pair on $\mathcal{F}$, $(\succsim', \succ')$ the extended precedence pair (introduced in Section 5), and $\mathcal{G}$ a set of function symbols that are flattened. Hereafter, we consider the signature $\mathcal{F} \cup \mathcal{F}_{\mathsf{c}}$ until the end of the section. For example, substitutions are those of terms over $\mathcal{F} \cup \mathcal{F}_{\mathsf{c}}$. Moreover, for brevity we omit the prime symbol $'$ from $\succsim'$ and $\succ'$.

The first key theorem states that $s \to_{\mathcal{R}} t$ implies $s{\downarrow} \succ_{\mathsf{mpo}} t{\downarrow}$, when $\mathcal{R} \subseteq \succ_{\mathcal{G}}$. To this end, we show that the relation $\succ_{\mathcal{G}}$ is closed under substitutions and flattening. The point is that, in contrast to MPOs, Definition 17 demands a greater term to be headed by a function symbol that is *not* flattened. For example, when $\sigma = \{x \mapsto \mathsf{a}\}$ and $\mathsf{f}^{(2)}, \mathsf{a}^{(0)} \in \mathcal{G}$, it holds that $\mathsf{f}(x, y) >_{\mathsf{mpo}} y$ by any MPO $>_{\mathsf{mpo}}$ but $\mathsf{f}(x, y)\sigma{\downarrow} = y = y\sigma{\downarrow}$.

▶ **Lemma 32.** *If $s \succ_{\mathcal{G}} t$ then $s\sigma \succ_{\mathcal{G}} t\sigma$ for all substitutions σ.*

Proof Sketch. Show $s\sigma \succ_{\mathsf{mpo}} t\sigma$ by induction on the derivation of $s \succ_{\mathcal{G}} t$. ◀

▶ **Lemma 33.** *If $s \succ_{\mathcal{G}} t$ then $s{\downarrow} \succ_{\mathsf{mpo}} t{\downarrow}$.*

The proof of Lemma 33 is in Appendix B.

The implication $s{\downarrow} \succ_{\mathcal{G}} t{\downarrow} \implies C[s]{\downarrow} \succ_{\mathcal{G}} C[t]{\downarrow}$ does not hold in general, as witnessed by $\mathsf{f}(x) \succ_{\mathcal{G}} x$ but $\mathsf{g}(\mathsf{f}(x)) \not\succ_{\mathcal{G}} \mathsf{g}(x)$ for $\mathcal{G} = \varnothing$. However, its super-relation $\succ_{\mathsf{mpo}}$ satisfies the corresponding property $s{\downarrow} \succ_{\mathsf{mpo}} t{\downarrow} \implies C[s]{\downarrow} \succ_{\mathsf{mpo}} C[t]{\downarrow}$. Note that $\succ_{\mathsf{mpo}}$ is closed under contexts but not flattening (consider $\mathsf{c}_1(x) \succ_{\mathsf{mpo}} x$). We prepare auxiliary lemmata.

▶ **Lemma 34.** *The inequality $t \succsim_{\mathsf{mpo}} t{\downarrow}$ holds for all terms t.*

Proof. It follows immediately from the fact that $\succsim_{\mathsf{mpo}}$ is a rewrite relation (Theorem 5) and $\ell \succsim_{\mathsf{mpo}} r$ holds for all $\ell \to r \in \mathsf{F}(\mathcal{G})$. ◀

We say that a term t is *root-rigid* if t is a variable or $t = f(t_1, \dots, t_n)$ with $f \in \mathcal{F} \setminus \mathcal{G}$. It is easy to see that if $t{\downarrow} = c_n(t_1, \dots, t_n)$ then $t_1, \dots, t_n$ are root-rigid.

▶ **Lemma 35.** *Let $C = f(u_1, \dots, u_{i-1}, \Box, u_{i+1}, \dots, u_n)$. If $s{\downarrow} \succ_{\mathsf{mpo}} t{\downarrow}$ then $C[s]{\downarrow} \succ_{\mathsf{mpo}} C[t]{\downarrow}$.*

Proof. Suppose $s{\downarrow} \succ_{\mathsf{mpo}} t{\downarrow}$. We show $C[s]{\downarrow} \succ_{\mathsf{mpo}} C[t]{\downarrow}$ by well-founded induction on $C[s]$ with respect to $\to^+_{\mathsf{F}(\mathcal{G})}$. If $f \in \mathcal{F} \setminus \mathcal{G}$ then the claim immediately follows by MPO (3). So assume $f \in \mathcal{G} \cup \mathcal{F}_{\mathsf{c}}$. The case when $C{\downarrow} = \Box$ is trivial. If C is $\mathsf{F}(\mathcal{G})$-reducible, we can construct a context C' such that $C[s] \to^+_{\mathsf{F}(\mathcal{G})} C'[s]$, and the induction hypothesis yields the desired inequality $C[s]{\downarrow} = C'[s]{\downarrow} \succ_{\mathsf{mpo}} C'[t]{\downarrow} = C[t]{\downarrow}$. Otherwise, C is already a flattened context, namely $C = \mathsf{c}_n(u_1, \dots, \Box, \dots, u_n)$ with $n \geqslant 2$ and $u_1, \dots, u_n$ root-rigid. An easy case is when $C[s]{\downarrow} = C[s{\downarrow}]$, which can be easily handled by Lemma 34 and closure under contexts of $\succ_{\mathsf{mpo}}$. The remaining case is when $s{\downarrow} = \mathsf{c}_m(s_1, \dots, s_m)$. Since c_0 is minimal with respect to $\succ_{\mathsf{mpo}}$ and $\mathsf{c}_1(s_1)$ is not flattened, we have $m \geqslant 2$, which leads to $m + n - 1 \geqslant 2$. Since $s_1, \dots, s_m$ are root-rigid, $C[s]{\downarrow} = \mathsf{c}_{m+n-1}(u_1, \dots, u_{i-1}, s_1, \dots, s_m, u_{i+1}, \dots, u_n)$ is obtained. We further distinguish three cases.

1. If $s{\downarrow} \succ_{\mathsf{mpo}} t{\downarrow}$ is derived by MPO (1), $s_i \succsim_{\mathsf{mpo}} t{\downarrow}$ holds for some $1 \leqslant i \leqslant m$. Because $m \geqslant 2$, we have $\{s_1, \dots, s_m\} \succ^{\mathsf{mul}}_{\mathsf{mpo}} \{t{\downarrow}\}$. So we obtain

$$\{u_1, \dots, u_{i-1}, s_1, \dots, s_m, u_{i+1}, \dots, u_n\} \succ^{\mathsf{mul}}_{\mathsf{mpo}} \{u_1, \dots, u_{i-1}, t{\downarrow}, u_{i+1}, \dots, u_n\}$$

   from which $C[s]{\downarrow} \succ_{\mathsf{mpo}} C[t{\downarrow}]$ follows by MPO (3).[1] Because $C[t{\downarrow}] \succsim_{\mathsf{mpo}} C[t]{\downarrow}$, the compatibility of $\succsim_{\mathsf{mpo}}$ and $\succ_{\mathsf{mpo}}$ entails the claim.
2. If $s{\downarrow} \succ_{\mathsf{mpo}} t{\downarrow}$ is derived by MPO (2), the root symbol of t is smaller than c_m. This contradicts the minimality of c_m.
3. If $s{\downarrow} \succ_{\mathsf{mpo}} t{\downarrow}$ is derived by MPO (3), $t{\downarrow}$ is of the form $\mathsf{c}_k(t_1, \dots, t_k)$ and the inequality $\{s_1, \dots, s_m\} \succ_{\mathsf{mpo}} \{t_1, \dots, t_k\}$ holds. So $C[s]{\downarrow} = \mathsf{c}_{n+m-1}(u_1, \dots, s_1, \dots, s_m, \dots, u_n)$ and $C[t]{\downarrow} = \mathsf{c}_{n+k-1}(u_1, \dots, t_1, \dots, t_k, \dots, u_n)$. Thus, $C[s]{\downarrow} \succ_{\mathsf{mpo}} C[t]{\downarrow}$ follows by MPO (3). ◀

▶ **Lemma 36.** *If $s{\downarrow} \succ_{\mathsf{mpo}} t{\downarrow}$ then $C[s]{\downarrow} \succ_{\mathsf{mpo}} C[t]{\downarrow}$ for all contexts C.*

Proof. The claim is shown by straightforward structural induction on C using Lemma 35. ◀

Combining these properties, we obtain the first key theorem.

▶ **Theorem 37.** *Let $\mathcal{R}$ be a TRS with $\mathcal{R} \subseteq \succ_{\mathcal{G}}$ and let s and t be terms. If $s \to_{\mathcal{R}} t$ then $s{\downarrow} \succ_{\mathsf{mpo}} t{\downarrow}$.*

Proof. Let $s \to_{\mathcal{R}} t$. There exist a rule $\ell \to r \in \mathcal{R}$, a substitution σ, and a context C such that $s = C[\ell\sigma]$ and $t = C[r\sigma]$. Since $\ell \succ_{\mathcal{G}} r$ holds by assumption, we obtain the implications:

$$\ell \succ_{\mathcal{G}} r \overset{32}{\Longrightarrow} \ell\sigma \succ_{\mathcal{G}} r\sigma \overset{33}{\Longrightarrow} \ell\sigma{\downarrow} \succ_{\mathsf{mpo}} r\sigma{\downarrow} \overset{36}{\Longrightarrow} C[\ell\sigma]{\downarrow} \succ_{\mathsf{mpo}} C[r\sigma]{\downarrow}$$

Here the numbers indicate the employed lemmata. Thus, $s{\downarrow} \succ_{\mathsf{mpo}} t{\downarrow}$ holds. ◀

The second key theorem states that $s \to_{\mathcal{S}} t$ implies $s{\downarrow} \succsim_{\mathsf{mpo}} t{\downarrow}$, provided that $\mathcal{S}$ is non-duplicating and $\ell \succsim_{\mathcal{G}} r$ or $r \in \mathcal{T}(\mathcal{G}, \mathcal{V})$ holds for all $\ell \to r \in \mathcal{S}$. To this end we show that, if $\ell \to r \in \mathcal{S}$ then $\ell\sigma{\downarrow} \succsim_{\mathsf{mpo}} r\sigma{\downarrow}$. The next lemma addresses the case when $s \to_{\mathcal{S}} t$ employs a rule $\ell \to r$ with $\ell \succsim_{\mathcal{G}} r$.

[1] For example, consider $C = \mathsf{c}_2(\Box, y)$, $s = \mathsf{c}_2(x, x)$, and $t = x$. The inequality $s{\downarrow} = \mathsf{c}_2(x, x) \succ_{\mathsf{mpo}} x = t{\downarrow}$ is derived by MPO (1), while $C[s]{\downarrow} = \mathsf{c}_3(x, x, y) \succ_{\mathsf{mpo}} \mathsf{c}_2(x, y) = C[t]{\downarrow}$ is derived by MPO (3).

▶ **Lemma 38.** *If $\ell \gtrsim_{\mathcal{G}} r$ then $\ell\sigma\downarrow \gtrsim_{\mathsf{mpo}} r\sigma\downarrow$ for all substitutions σ.*

Proof. The proof is analogous to Lemma 32 and Lemma 33. ◀

The other case when $s \to_{\mathcal{S}} t$ uses a rule $\ell \to r$ with $r \in \mathcal{T}(\mathcal{G}, \mathcal{V})$ is more difficult. Actually $s\downarrow \gtrsim_{\mathsf{mpo}} t\downarrow$ does not hold in general. Consider the case that $s = \ell = \mathsf{f}(x, \mathsf{a})$ and $t = r = \mathsf{a}$ with $\mathsf{f}, \mathsf{a} \in \mathcal{G}$. Because $s\downarrow = x$ and $t\downarrow = \mathsf{c}_0$, the inequality $s\downarrow \gtrsim_{\mathsf{mpo}} t\downarrow$ does not hold. Fortunately, the claim holds when s is ground.

▶ **Lemma 39.** *If s is ground and $s \trianglerighteq t$ then $s\downarrow \gtrsim_{\mathsf{mpo}} t\downarrow$.*

The proof is in Appendix B. It is essential for Lemma 39 that s is ground, as seen by the example above.

▶ **Lemma 40.** *Let $t = f(t_1, \dots, t_n)$. If $\mathcal{F}\mathsf{un}(t) \cap \mathcal{G} = \varnothing$ and $t \in \mathsf{NF}(\xrightarrow{\epsilon}_{\mathsf{F}(\mathcal{G})})$ then $t\downarrow = f(t_1\downarrow, \dots, t_n\downarrow)$.* ◀

▶ **Lemma 41.** *Let (s, t) be a non-duplicating pair with $t \in \mathcal{T}(\mathcal{G} \cup \mathcal{F}_{\mathsf{c}}, \mathcal{V})$ and σ a grounding substitution for s and t. The relation $s\sigma\downarrow \gtrsim_{\mathsf{mpo}} t\sigma\downarrow$ holds.*

Proof. We show the claim by induction on $s\sigma$ with respect to $\to^+_{\mathsf{F}(\mathcal{G})}$. The case when $t\downarrow$ is a variable is routine. So suppose $t\downarrow = \mathsf{c}_n(x_1, \dots, x_n)$. Depending on reducibility of $s\sigma$ by $\mathsf{F}(\mathcal{G})$, we distinguish several cases.

1. If s is not flattened then $s\sigma \to^+_{\mathsf{F}(\mathcal{G})} s\downarrow\sigma$. Since $|s\downarrow|_x = |s|_x \geqslant |t|_x$ for all variables x, the induction hypothesis yields $s\sigma\downarrow = (s\downarrow\sigma)\downarrow \gtrsim_{\mathsf{mpo}} t\sigma\downarrow$.
2. If $x\sigma$ is not flattened for some $x \in \mathcal{V}\mathsf{ar}(s)$ then $s\sigma \to^+_{\mathsf{F}(\mathcal{G})} s\tau$, where τ is given by $\tau(y) = y\sigma\downarrow$ for each variable y. By the induction hypothesis $s\sigma\downarrow = s\tau\downarrow \gtrsim_{\mathsf{mpo}} t\tau\downarrow = t\sigma\downarrow$ follows.
3. If the last two conditions are not satisfied and $s\sigma \notin \mathsf{NF}(\xrightarrow{\epsilon}_{\mathsf{F}(\mathcal{G})})$ then $s = \mathsf{c}_m(s_1, \dots, s_m)$ with s_i a variable and $s_i\sigma\downarrow = \mathsf{c}_k(u_1, \dots, u_k)$. Recall $t\downarrow = \mathsf{c}_n(x_1, \dots, x_n)$.
   - If $s_i = x_j$ for some $1 \leqslant j \leqslant n$ then consider the terms

     $$s' = \mathsf{c}_{m+k-1}(s_1, \dots, s_{i-1}, y_1, \dots, y_k, s_{i+1}, \dots, s_m)$$
     $$t' = \mathsf{c}_{n+k-1}(x_1, \dots, x_{j-1}, y_1, \dots, y_k, x_{j+1}, \dots, x_n)$$

     and the substitution τ given by $\tau(y_h) = u_h$ for $h \in \{1, \dots, k\}$ and $\tau(z) = \sigma(z)$ for other variables z, where $y_1, \dots, y_k$ are fresh variables. Since $s\sigma \to^+_{\mathsf{F}(\mathcal{G})} s'\tau$ and $|s'|_x \geqslant |t'|_x$ for all variables x, by the induction hypothesis we obtain $s\sigma\downarrow = s'\tau\downarrow \gtrsim_{\mathsf{mpo}} t'\tau\downarrow = t\sigma\downarrow$.
   - Otherwise, consider the term $s' = \mathsf{c}_{m+k-1}(s_1, \dots, s_{i-1}, u_1, \dots, u_k, s_{i+1}, \dots, s_m)$. We have $s\sigma \to^+_{\mathsf{F}(\mathcal{G})} s'\sigma$ and $|s'|_x \geqslant |t|_x$ for all variables x. By the induction hypothesis $s\sigma\downarrow = s'\sigma\downarrow \gtrsim_{\mathsf{mpo}} t\sigma\downarrow$ follows.
4. Otherwise, s and $x\sigma$ for all $x \in \mathcal{V}\mathsf{ar}(s)$ are flattened, and $s\sigma \in \mathsf{NF}(\xrightarrow{\epsilon}_{\mathsf{F}(\mathcal{G})})$. Since the former condition guarantees $\mathcal{F}\mathsf{un}(s\sigma) \cap \mathcal{G} = \varnothing$, Lemma 40 yields $s\sigma\downarrow = f(s_1\sigma\downarrow, \dots, s_m\sigma\downarrow)$. Recalling $t\downarrow = \mathsf{c}_n(x_1, \dots, x_n)$, we further distinguish two cases.
   (a) If $f \in \mathcal{F} \setminus \mathcal{G}$ then $f \succ \mathsf{c}_n$. In addition, $s\downarrow \succ_{\mathcal{G}} x_j$ holds for all j. Since $\succ_{\mathcal{G}}$ is closed under substitutions and flattening (Lemmata 32 and 33), $s\sigma\downarrow \succ_{\mathcal{G}} x_j\sigma\downarrow$ is obtained. Thus, $s\sigma\downarrow \succ_{\mathsf{mpo}} x_j\sigma\downarrow$. Hence, MPO (2) entails the claim.
   (b) Otherwise, $f = \mathsf{c}_m$ and $s_i\sigma$ are root-rigid for all i. Let Y_i denote the multiset of variables in s_i. Since $|s|_x \geqslant |t|_x$ for all variables x, the multiset $\{x_1, \dots, x_n\}$ can be represented by $X_1 \uplus \cdots \uplus X_m$ with $X_i \subseteq Y_i$ for each $1 \leqslant i \leqslant m$. First we show the subgoal $\{s_i\sigma\downarrow\} \gtrsim^{\mathsf{mul}}_{\mathsf{mpo}} \{x\sigma\downarrow \mid x \in X_i\}$:
   - If $X_i = \varnothing$ then the claim is trivial.

- If $X_i = \{x\}$ for some variable x then $s_i \trianglerighteq x$, and thus $s_i\sigma \trianglerighteq x\sigma$. By Lemma 39 the claim follows.
- Otherwise, $|Y_i| \geqslant |X_i| \geqslant 2$. As s_i contains at least two variables, s_i is not a variable. So for every $x \in X_i$ we have $s_i \rhd x$. As $\mathsf{root}(s_i) \in \mathcal{F} \setminus \mathcal{G}$, the relation $s_i \succ_{\mathcal{G}} x$ holds. As in Case (a), we can deduce $s_i\sigma{\downarrow} \succ_{\mathsf{mpo}} x\sigma{\downarrow}$. Thus, the claim holds.

The subgoal results in $\{s_1\sigma{\downarrow}, \dots, s_m\sigma{\downarrow}\} \succsim^{\mathsf{mul}} \{x_1\sigma{\downarrow}, \dots, x_n\sigma{\downarrow}\}$. Thus, the inequalities

$$s\sigma{\downarrow} = \mathsf{c}_m(s_1\sigma{\downarrow}, \dots, s_m\sigma{\downarrow}) \succsim_{\mathsf{mpo}} \mathsf{c}_n(x_1\sigma{\downarrow}, \dots, x_n\sigma{\downarrow}) \succsim_{\mathsf{mpo}} t\sigma{\downarrow}$$

are obtained by MPO (3) and Lemma 34. ◀

As in the case of $\succ_{\mathsf{mpo}}$ (Lemmata 35 and 36), one can verify that $\succsim_{\mathsf{mpo}}$ is preserved under the combination of context application and flattening.

▶ **Lemma 42.** *If $s{\downarrow} \succsim_{\mathsf{mpo}} t{\downarrow}$ then $C[s]{\downarrow} \succsim_{\mathsf{mpo}} C[t]{\downarrow}$ for all contexts C.* ◀

We arrive at the second key theorem for $\mathcal{S}$-steps.

▶ **Theorem 43.** *Let $\mathcal{S}$ be a non-duplicating TRS such that $\ell \succsim_{\mathcal{G}} r$ or $r \in \mathcal{T}(\mathcal{G}, \mathcal{V})$ for all $\ell \to r \in \mathcal{S}$. Let s and t be ground terms. If $s \to_{\mathcal{S}} t$ then $s{\downarrow} \succsim_{\mathsf{mpo}} t{\downarrow}$.*

Proof. Let $s \to_{\mathcal{S}} r$. There exist a rule $\ell \to r \in \mathcal{S}$, a grounding substitution σ for s and t, and a context C such that $s = C[\ell\sigma]$ and $t = C[r\sigma]$. We have the following implications:

$$\ell \to r \in \mathcal{S} \implies \ell\sigma{\downarrow} \succsim_{\mathsf{mpo}} r\sigma{\downarrow} \overset{42}{\implies} C[\ell\sigma]{\downarrow} \succ_{\mathsf{mpo}} C[r\sigma]{\downarrow}$$

The first implication follows from Lemma 38 or Lemma 41. Thus, $s{\downarrow} \succsim_{\mathsf{mpo}} t{\downarrow}$ holds. ◀

Theorem 30 is a consequence of Theorems 37 and 43.

Proof of Theorem 30. It suffices to show termination of $\mathcal{R}/\mathcal{S}$ under the extended signature $\mathcal{F} \cup \mathcal{F}_{\mathsf{c}}$. Since $\mathcal{F}_{\mathsf{c}}$ contains the constant c_0, every infinite rewrite sequence of terms can turn into an infinite rewrite sequence of ground terms by instantiating variables to c_0. Therefore, our task boils down to proving termination on ground terms. Consider ground terms s and t. By Theorems 37 and 43 we obtain the implications:

$$s \to_{\mathcal{R}/\mathcal{S}} t \implies s \to_{\mathcal{S}}^* \cdot \to_{\mathcal{R}} \cdot \to_{\mathcal{S}}^* t \implies s{\downarrow} \succsim_{\mathsf{mpo}} \cdot \succ_{\mathsf{mpo}} \cdot \succsim_{\mathsf{mpo}} t{\downarrow} \implies s{\downarrow} \succ_{\mathsf{mpo}} t{\downarrow}$$

As $\succ_{\mathsf{mpo}}$ is well-founded, $\to_{\mathcal{R}/\mathcal{S}}$ is terminating on ground terms. ◀

7 Improving Applicability

Theorem 23 is a simple and elegant adaptation of the original method (Theorem 11). However, the dominance condition can be a severe restriction, for example, in confluence analysis based on relative termination.

▶ **Example 44.** Consider the confluence problem of the following TRS:

1:	$x + \mathsf{s}(y) \to \mathsf{s}(x) + y$	6:	$x \times \mathsf{s}(y) \to x + (x \times y)$
2:	$\mathsf{s}(x) + y \to x + \mathsf{s}(y)$	7:	$\mathsf{s}(x) \times y \to (x \times y) + y$
3:	$x + y \to y + x$	8:	$x \times y \to y \times x$
4:	$(x + y) + z \to x + (y + z)$	9:	$\mathsf{sq}(x) \to x \times x$
5:	$x + (y + z) \to (x + y) + z$	10:	$\mathsf{sq}(\mathsf{s}(x)) \to (x \times x) + \mathsf{s}(x + x)$

By the rule labeling technique by Zankl et al. (see [28, Example 17]) the confluence problem boils down to the relative termination problem of $\mathcal{R}/\mathcal{S}$, where $\mathcal{R}$ is the set of all duplicating rules in the above TRS and $\mathcal{S}$ is the set of the non-duplicating rules; namely $\mathcal{R} = \{6, 7, 9, 10\}$ and $\mathcal{S} = \{1, 2, 3, 4, 5, 8\}$. In order to use Theorem 23 it is necessary that $\mathcal{S}$ is non-duplicating and $\mathcal{R}$ dominates $\mathcal{S}$. The former is satisfied, but the latter does not hold. In fact, $\mathcal{D}_{\mathcal{R}} = \{\times, \mathsf{sq}\}$ and $\times$ occurs in the right-hand side of rule 8. So Theorem 23 is not applicable to this example.

Multiset path orders are capable of dealing with termination modulo commutative (or permutative) axioms, where Theorem 23 fails due to absence of dominance. Using our simulation technique, we incorporate this advantage into the dependency pair method for relative termination.

We introduce a generalized notion of dominance. We say that a rule is *transitional* if it is of the form $f(x_1, \dots, x_n) \to f(y_1, \dots, y_n)$ with $x_1, \dots, x_n, y_1, \dots, y_n$ variables. Note that the variables need not be different from each other.

▶ **Definition 45.** *Let $\mathcal{R}$ and $\mathcal{S}$ be TRSs over the same signature and $\mathcal{G} = \mathcal{C}_{\mathcal{R}}$. We say that $\mathcal{R}$* almost dominates *$\mathcal{S}$ if for every $\ell \to r \in \mathcal{S}$ either $r \in \mathcal{T}(\mathcal{C}_{\mathcal{R}}, \mathcal{V})$ or $\ell \to r{\downarrow_{\mathcal{G}}}$ is a transitional rule with $\mathsf{root}(\ell) \in \mathcal{D}_{\mathcal{R}}$.*

We incorporate $\mathsf{DP}_{\mathcal{D}_{\mathcal{R}}}(\mathcal{S})$ (see Definition 10) into Theorem 23. We denote this set by $\mathsf{DP}(\mathcal{S})$. When a rule $\ell \to r \in \mathcal{S}$ satisfies the second condition in Definition 45, it gives rise to exactly one dependency pair.

▶ **Theorem 46.** *Suppose that a TRS $\mathcal{R}$ almost dominates a non-duplicating TRS $\mathcal{S}$. Then $\mathcal{R}/\mathcal{S}$ is terminating if there is a reduction pair $(\gtrsim, >)$ with $\mathsf{DP}(\mathcal{R}) \subseteq >$ and $\mathcal{R} \cup \mathcal{S} \cup \mathsf{DP}(\mathcal{S}) \subseteq \gtrsim$.*

▶ Remark 47. As discussed in Remark 12, if Theorem 46 is applicable, relative termination follows from finiteness of $(\mathsf{DP}(\mathcal{R}), \mathcal{R} \cup \mathcal{S} \cup \mathsf{DP}(\mathcal{S}))$. So all methods for the dependency pair framework [13], including the iterative use of reduction pairs [14] and dependency graph techniques [4], are available for showing relative termination.

For proving the theorem we need an improved version of Theorem 30.

▶ **Theorem 48.** *Let $\mathcal{R}$ and $\mathcal{S}$ be TRSs over a signature $\mathcal{F}$. Then $\mathcal{R}/\mathcal{S}$ is terminating if there exist a well-founded precedence pair $(\succsim, \succ)$ and a subset $\mathcal{G}$ of $\mathcal{F}$ such that $\mathcal{R} \subseteq \succ_{\mathcal{G}}$, $\mathcal{S}$ is non-duplicating, and every $\ell \to r \in \mathcal{S}$ satisfies one of the following alternatives:*

1. *$\ell \succsim_{\mathcal{G}} r$*
2. *$r \in \mathcal{T}(\mathcal{G}, \mathcal{V})$*
3. *$\ell = f(x_1, \dots, x_m) \succsim_{\mathsf{mpo}} g(y_1, \dots, y_n) = r{\downarrow_{\mathcal{G}}}$ for some variables $x_1, \dots, x_m, y_1, \dots, y_n$ and $f, g \in \mathcal{F} \setminus \mathcal{G}$*

Here $\succ_{\mathcal{G}}$, $\succsim_{\mathcal{G}}$ and $\succsim_{\mathsf{mpo}}$ are the relations induced by the precedence pair $(\succsim, \succ)$.

Proof. The proof is analogous to that of Theorem 30, but we need to extend Theorem 43 to cover the case when $\ell \to r \in \mathcal{S}$ satisfies the third condition. This is archived by showing $\ell\sigma{\downarrow} \succsim_{\mathsf{mpo}} r\sigma{\downarrow}$ for all substitutions σ. The inequality is verified by easy case distinction. ◀

Proof of Theorem 46. The proof follows the simulation result of Section 5. Given a reduction pair $(\geqslant_{\mathcal{A}}, >_{\mathcal{A}})$, we label $\mathcal{R}$ and $\mathcal{S}$ with the $\mathcal{A}$-induced labeling. By taking the $\mathcal{A}$-induced precedence and $\mathcal{G} = \mathcal{C}_{\mathcal{R}}$ termination of $\mathcal{R}_{\mathsf{lab}}/(\mathcal{S}_{\mathsf{lab}} \cup \mathcal{D}\mathsf{ec}(\geqslant_{\mathcal{A}}))$ follows from Theorem 48. Hence the claim holds. Note that $\mathsf{DP}(\mathcal{S}) \subseteq \geqslant_{\mathcal{A}}$ guarantees $f_a \succsim g_b$ in the case of transitional rules in Definition 45, and therefore handled by the third case of Theorem 48. ◀

▶ **Example 49** (continued from Example 44)**.** Now we switch from Theorem 23 to Theorem 46. Recalling $\mathcal{D}_{\mathcal{R}} = \{\times, \mathsf{sq}\}$, we can easily see that $\mathcal{R}$ almost dominates $\mathcal{S}$. The set $\mathsf{DP}(\mathcal{R})$ of dependency pairs consists of the four rules

$$x \times^\sharp \mathsf{s}(y) \to x \times^\sharp y \qquad \mathsf{s}(x) \times^\sharp y \to x \times^\sharp y \qquad \mathsf{sq}^\sharp(x) \to x \times^\sharp x \qquad \mathsf{sq}^\sharp(\mathsf{s}(x)) \to x \times^\sharp x$$

and $\mathsf{DP}(\mathcal{S}) = \{x \times^\sharp y \to y \times^\sharp x\}$. Take the following weakly monotone algebra $\mathcal{A}$ on $\mathbb{N}$:

$$\mathsf{s}_{\mathcal{A}}(x) = x + 1 \qquad x +_{\mathcal{A}} y = 0 \qquad x \times_{\mathcal{A}} y = x \times^\sharp_{\mathcal{A}} y = x + y \qquad \mathsf{sq}_{\mathcal{A}}(x) = \mathsf{sq}^\sharp_{\mathcal{A}}(x) = 2x + 1$$

The reduction pair $(\geqslant_{\mathcal{A}}, >_{\mathcal{A}})$ satisfies $\mathsf{DP}(\mathcal{R}) \subseteq >_{\mathcal{A}}$ and $\mathcal{R} \cup \mathcal{S} \cup \mathsf{DP}(\mathcal{S}) \subseteq \geqslant_{\mathcal{A}}$, and therefore $\mathcal{R}/\mathcal{S}$ is terminating. Observe that the proof via Theorem 46 only uses linear polynomials, but a termination proof of $\mathcal{R}/\mathcal{S}$ by polynomial interpretation over $\mathbb{N}$ demands quadratic ones.

8 Conclusion

We conclude the paper by discussing experimental results and related work.

Evaluation by experiments. In order to assess practicality we have implemented a prototype tool for relative termination based on the improved dependency pair method (Theorem 46).[2] Following Remark 47, the tool attempts to prove finiteness of the corresponding dependency pair problem by iterative application of reduction pairs based on ordinal interpretations below ω^3 as in Example 24. The tool shows relative termination of 48 problems in the `TRS_Relative` category of TPDB 11.3 [24], which consists of 98 problems. There are 8 problems that satisfy the relaxed preconditions (non-duplicatingness and almost dominance) of Theorem 46 but not dominance of Theorem 23 due to Iborra et al. Among them, 6 problems are proved terminating. While all the 6 problems are solved by at least one of existing tools, the use of the dependency pair method (Theorem 46) often makes proofs easier. For example, the problem `Relative_05/rt2-1` asks to show the relative termination of $\{\mathsf{T}(\mathsf{I}(x), y) \to \mathsf{T}(x, y)\}$ with respect to $\{\mathsf{T}(x, y) \to \mathsf{T}(x, \mathsf{I}(y))\}$. Since the almost dominance condition holds, the dependency pair method with the linear polynomial interpretation $\mathsf{T}_{\mathcal{A}}(x, y) = \mathsf{T}^\sharp_{\mathcal{A}}(x, y) = x$ and $\mathsf{I}_{\mathcal{A}}(x) = x + 1$ proves the relative termination. In contrast, the 2023 version of AProVE and $\mathsf{TTT2}$ solve this problem by using two- and five-dimensional matrix interpretations, respectively.

Correctness proofs for dependency pairs. Using a model-based version of semantic labeling [29, Theorem 4], Arts showed correctness of an earlier version of the dependency pair method [3, Theorem 9]. In contrast to Theorem 21, this proof is involved and restricted to constructor TRSs. Later, the proof was simplified by switching to a direct proof based on the notion of minimal non-terminating term [4]. This became the standard proof method. In the presented work we re-introduced semantic labeling. A key difference is that our work adopts the one based on quasi-models ([29, Theorem 8] and [12, Corollary 1]).

Potential future work is to extend the presented simulation methodology to dependency pair methods for other rewriting formats. We anticipate that, with suitable semantic labeling, AC-dependency pairs [1, 27] can be simulated by Rubio's AC-RPO [21] and that dependency

[2] The tool and the experimental data (including comparison to existing termination tools) are available at: `https://www.jaist.ac.jp/project/saigawa/24fscd/`.

pairs based on strong computability [18, 10] can be simulated by higher-order RPO [16]. This is not only of theoretical interest, since thus-obtained proofs might ease formalization in proof assistants, provided signatures extension (caused by labeling) can be smoothly handled, see [2, 7, 22] for related discussions.

Completeness of semantic labeling and precedence termination. It is known that semantic labeling with precedence termination (cf. Proposition 19) is a complete method for showing termination of TRSs [19, Theorem 4], meaning that if a TRS is terminating then the termination is shown by semantic labeling and a simpler version of precedence termination. We remark that the combination of Theorems 11 and 21 yields a similar result. As for relative termination, the combination of Theorems 23 and 31 shows completeness of semantic labeling for TRSs with dominance and non-duplicatingness. It is future work to extend this result to a wider class of TRSs.

References

1 B. Alarcón, S. Lucas, and J. Meseguer. A dependency pair framework for A∨C-termination. In *Proceedings of the 8th International Workshop on Rewriting Logic and its Applications*, LNCS, pages 36–52, 2010. doi:10.1007/978-3-642-16310-4_4.

2 A. Almeida and M. Ayala-Rincón. Formalizing the dependency pair criterion for innermost termination. *Science of Computer Programming*, 195:102474, 2020. doi:10.1016/j.scico.2020.102474.

3 T. Arts. Termination by absence of infinite chains of dependency pairs. In *Proc. 21st International Colloquium on Trees in Algebra and Programming*, volume 1059 of *LNCS*, pages 196–210, 1996. doi:10.1007/3-540-61064-2_38.

4 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.

5 F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

6 L. Bachmair and D. Plaisted. Associative path orderings. In *Proc. 1st International Conference on Rewriting Techniques and Applications*, volume 202 of *LNCS*, pages 241–254, 1985. doi:10.1007/3-540-15976-2_11.

7 F. Blanqui and A. Koprowski. CoLoR: a Coq library on well-founded rewrite relations and its application to the automated verification of termination certificates. *Mathematical Structures in Computer Science*, 21(4):827–859, 2011. doi:10.1017/S0960129511000120.

8 N. Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science*, 17:279–301, 1982. doi:10.1016/0304-3975(82)90026-3.

9 N. Dershowitz. Termination by abstraction. In *Proc. 20th International Conference on Logic Programming*, volume 3132 of *LNCS*, pages 1–18, 2004. doi:10.1007/978-3-540-27775-0_1.

10 C. Fuhs and C. Kop. A static higher-order dependency pair framework. In *Proceedings of ESOP 2019*, volume 11423 of *LNCS*, pages 752–782, 2019. doi:10.1007/978-3-030-17184-1_27.

11 A. Geser. *Relative Termination*. PhD thesis, Universität Passau, 1990. URL: http://vts.uni-ulm.de/docs/2012/8146/vts_8146_11884.pdf.

12 A. Geser. Semantic labelling followed by recursive path order with status is strictly more powerful than the corresponding semantic path order. Technical report, Universität Passau, 1994.

13 J. Giesl, R. Thiemann, and P. Schneider-Kamp. The dependency pair framework: Combining techniques for automated termination proofs. In *Proc. 11th International Conference on Logic Programming and Automated Reasoning*, volume 3452 of *LNCS (LNAI)*, pages 301–331, 2005. doi:10.1007/978-3-540-32275-7_21.

14 Nao Hirokawa and Aart Middeldorp. Automating the dependency pair method. *Information and Computation*, 199:172–199, 2005. doi:10.1016/j.ic.2004.10.004.

15 J. Iborra, N. Nishida, G. Vidal, and A. Yamada. Relative termination via dependency pairs. *Journal of Automated Reasoning*, 58:391–411, 2017. doi:10.1007/s10817-016-9373-5.

16 J.-P. Jouannaud and A. Rubio. The higher-order recursive path ordering. In *Proc. 14th Symposium on Logic in Computer Science*, pages 402–411, 1999. doi:10.1109/LICS.1999.782635.

17 S. Kamin and J.J. Lévy. Two generalizations of the recursive path ordering. Technical report, University of Illinois, 1980. Unpublished manuscript.

18 K. Kusakari, Y. Isogai, M. Sakai, and F. Blanqui. Static dependency pair method based on strong computability for higher-order rewrite systems. *IEICE Transactions on Information and Systems*, E92.D(10):2007–2015, 2009. doi:10.1587/transinf.E92.D.2007.

19 A. Middeldorp, H. Ohsaki, and H. Zantema. Transforming termination by self-labelling. In *Proc. 13th International Conference on Automated Deduction*, volume 1104 of *LNCS*, pages 373–387, 1996. doi:10.1007/3-540-61511-3_101.

20 H. Ohsaki, A. Middeldorp, and J. Giesl. Equational termination by semantic labelling. In *Proc. 14th Annual Conference of the European Association for Computer Science Logic*, volume 1862 of *LNCS*, pages 457–471, 2000. doi:10.1007/3-540-44622-2_31.

21 Albert Rubio. A fully syntactic AC-RPO. *Information and Computation*, 178(2):515–533, 2002. doi:10.1006/inco.2002.3158.

22 C. Sternagel and R. Thiemann. Signature extensions preserve termination - an alternative proof via dependency pairs. In *Proc. 19th Annual Conference of the European Association for Computer Science Logic*, volume 6247 of *LNCS*, pages 514–528, 2010.

23 Terese. *Term Rewriting Systems*. Cambridge University Press, 2003.

24 Termination Community. The Termination Problem Database (TPDB). https://github.com/TermCOMP/TPDB. Accessed: 05.12.2023.

25 R. Thiemann, G. Allais, and J. Nagele. On the formalization of termination techniques based on multiset orderings. In *Proc. 23rd International Conference on Rewriting Techniques and Applications*, volume 15 of *LIPIcs*, pages 339–354, 2012. doi:10.4230/LIPIcs.RTA.2012.339.

26 R. Thiemann, J. Schöpf, C. Sternagel, and A. Yamada. Certifying the weighted path order. In *Proc. 5th International Conference on Formal Structures on Computation and Deduction*, volume 167 of *LIPIcs*, pages 4:1–4:20, 2020. doi:10.4230/LIPICS.FSCD.2020.4.

27 A. Yamada, C. Sternagel, R. Thiemann, and K. Kusakari. AC dependency pairs revisited. In *Proc. 25th Annual Conference of the European Association for Computer Science Logic*, volume 62 of *LIPIcs*, pages 8:1–8:16, 2016. doi:10.4230/LIPICS.CSL.2016.8.

28 H. Zankl, B. Felgenhauer, and A. Middeldorp. Labelings for decreasing diagrams. *Journal of Automated Reasoning*, 54(2):101–133, 2015. doi:10.1007/S10817-014-9316-Y.

29 H. Zantema. Termination of term rewriting by semantic labelling. *Fundamenta Informaticae*, 24:89–105, 1995. doi:10.3233/FI-1995-24124.

A Proof of Geser's Semantic Labeling

To make the paper self-contained, we prove Theorem 7 by recasting Geser's original proof [12], see also [23, Section 6.5.4]. Let $(\mathcal{A}, \gtrsim)$ be a weakly monotone algebra and $\mathcal{L}$ a weakly monotone labeling for $(\mathcal{A}, \gtrsim)$. Suppose that TRSs $\mathcal{R}$ and $\mathcal{S}$ satisfy $\mathcal{R} \cup \mathcal{S} \subseteq \gtrsim_{\mathcal{A}}$.

▶ **Lemma 50** ([23, Lemma 6.5.31]). *The identity* $\mathsf{lab}(t\sigma, \alpha) = \mathsf{lab}(t, \beta)\tau$ *holds for all terms* t*, substitutions* σ*, and assignments* α*. Here* β *and* τ *are the assignment and the substitution defined by* $\beta(x) = [\alpha]_{\mathcal{A}}(x\sigma)$ *and* $\tau(x) = \mathsf{lab}(x\sigma, \alpha)$ *for all variables* x*.* ◀

▶ **Lemma 51** ([12, Theorem 2]). *If* $s \to_{\mathcal{R}} t$ *then* $\mathsf{lab}(s, \alpha) \to_{\mathcal{R}_{\mathsf{lab}}/\mathcal{D}\mathsf{ec}(\gtrsim)} \mathsf{lab}(t, \alpha)$ *for all assignments* α*.*

Proof. We show the statement by induction on $|s| + |t|$. Let $s \to_{\mathcal{R}} t$ and let α be an assignment. Depending on the rewrite position of $s \to_{\mathcal{R}} t$, we distinguish two cases.

- If the rewrite position is root then there exist $\ell \to r \in \mathcal{R}$ and a substitution σ such that $s = \ell\sigma$ and $t = r\sigma$. From Lemma 50 we obtain $\mathsf{lab}(s,\alpha) \to_{\mathcal{R}_{\mathsf{lab}}} \mathsf{lab}(t,\alpha)$ as follows:

$$\mathsf{lab}(s,\alpha) = \mathsf{lab}(\ell\sigma,\alpha) = \mathsf{lab}(\ell,\beta)\tau \to_{\mathcal{R}_{\mathsf{lab}}} \mathsf{lab}(r,\beta)\tau = \mathsf{lab}(r\sigma,\alpha) = \mathsf{lab}(t,\alpha)$$

- Suppose that $s = f(s_1,\dots,s_i,\dots,s_n)$ and the i-th argument s_i is rewritten. In this case, we have $t = f(s_1,\dots,t',\dots,s_n)$ and $s_i \to_{\mathcal{R}} t'$. The induction hypothesis yields $\mathsf{lab}(s_i,\alpha) \to_{\mathcal{R}_{\mathsf{lab}}/\mathcal{D}\mathsf{ec}(\gtrsim)} \mathsf{lab}(t',\alpha)$. Let $a = \mathsf{lab}_f([\alpha]_{\mathcal{A}}(s_1),\dots,[\alpha]_{\mathcal{A}}(s_i),\dots,[\alpha]_{\mathcal{A}}(s_n))$ and $b = \mathsf{lab}_f([\alpha]_{\mathcal{A}}(s_1),\dots,[\alpha]_{\mathcal{A}}(t'),\dots,[\alpha]_{\mathcal{A}}(s_n))$. From $\mathcal{R} \subseteq \gtrsim_{\mathcal{A}}$ and that $\gtrsim_{\mathcal{A}}$ is a rewrite preorder, we have $s_i \gtrsim_{\mathcal{A}} t'$. Moreover, since $\mathcal{L}$ is weakly monotone, the inequality $a \gtrsim b$ holds. So $f_a(x_1,\dots,x_n) \to f_b(x_1,\dots,x_n) \in \mathcal{D}\mathsf{ec}(\gtrsim)$. Finally, we obtain the claim as follows:

$$\begin{aligned}
\mathsf{lab}(s,\alpha) = \quad & f_a(\mathsf{lab}(s_1,\alpha),\dots,\mathsf{lab}(s_i,\alpha),\dots,\mathsf{lab}(s_n,\alpha)) \\
\to_{\mathcal{R}_{\mathsf{lab}}/\mathcal{D}\mathsf{ec}(\gtrsim)} \quad & f_a(\mathsf{lab}(s_1,\alpha),\dots,\mathsf{lab}(t',\alpha)\dots,\mathsf{lab}(s_n,\alpha)) \\
\to_{\mathcal{D}\mathsf{ec}(\gtrsim)} \quad & f_b(\mathsf{lab}(s_1,\alpha),\dots,\mathsf{lab}(t',\alpha),\dots,\mathsf{lab}(s_n,\alpha)) \\
= \quad & \mathsf{lab}(t,\alpha)
\end{aligned}$$

  This case concludes the proof. ◀

▶ **Lemma 52.** *If $s \to_{\mathcal{S}} t$ then $\mathsf{lab}(s,\alpha) \to_{\mathcal{S}_{\mathsf{lab}}/\mathcal{D}\mathsf{ec}(\gtrsim)} \mathsf{lab}(t,\alpha)$ for all assignments α.*

Proof. The same argument as Lemma 51 goes through. ◀

Proof of Theorem 7. The "if" direction follows from Lemmata 51 and 52. In order to show the "only if" direction we consider the contraposition. Suppose that $\mathcal{R}_{\mathsf{lab}}/(\mathcal{S}_{\mathsf{lab}} \cup \mathcal{D}\mathsf{ec}(\gtrsim))$ has an infinite rewrite sequence. Unlabeling each term in the sequence, we obtain an infinite rewrite sequence of $\mathcal{R}/\mathcal{S}$. ◀

B Omitted Proofs in Section 6

Lemma 33 claims that if $s \succ_{\mathcal{G}} t$ then $s{\downarrow} \succ_{\mathsf{mpo}} t{\downarrow}$. This follows from that $s \trianglerighteq t$ and $\mathsf{root}(t) \in \mathcal{F} \setminus \mathcal{G}$ imply $s{\downarrow} \trianglerighteq t{\downarrow}$.

▶ **Lemma 53.** *If $\mathsf{root}(t) \in \mathcal{F} \setminus \mathcal{G}$, $s \to_{\mathsf{F}(\mathcal{G})} s'$, and $s \trianglerighteq t$ then $\mathsf{root}(t') \in \mathcal{F} \setminus \mathcal{G}$, $t \to^{=}_{\mathsf{F}(\mathcal{G})} t'$, and $s' \trianglerighteq t'$ for some t'.*

Proof. Suppose $s \xrightarrow{p}_{\mathsf{F}(\mathcal{G})} s'$ and $s \trianglerighteq t$. We perform induction on p.

- If $s = t$ then $p > \epsilon$ because of $\mathsf{root}(t) \in \mathcal{F} \setminus \mathcal{G}$. By taking $t' = s'$ the claim holds.
- If $p = \epsilon$ then $s = \ell\sigma$ and $s' = r\sigma$ for some rule $\ell \to r \in \mathsf{F}(\mathcal{G})$. By assumption we have $\mathsf{root}(t) \notin \mathcal{F}\mathsf{un}(\ell)$, so there exists $x \in \mathcal{V}\mathsf{ar}(\ell)$ such that $x\sigma \trianglerighteq t$. Since $\mathcal{V}\mathsf{ar}(\ell) = \mathcal{V}\mathsf{ar}(r)$ holds, we obtain $s' \trianglerighteq x\sigma$.
- Otherwise, s is of form $f(s_1,\dots,s_n)$, $s_i \trianglerighteq t$, and $p = jq$ for some indexes $1 \leqslant i,j \leqslant n$ and position q.
  - If $i = j$ then by the induction hypothesis we obtain $t \to^{=}_{\mathsf{F}(\mathcal{G})} t'$ and $s'|_i \trianglerighteq t'$ for some t'.
  - If $i \neq j$ then by setting $t' = t$ we have $t \to^{=}_{\mathsf{F}(\mathcal{G})} t'$ and $s'|_i = s_i \trianglerighteq t'$.

  In either case t' is a subterm of s'. Therefore, the claim holds. ◀

▶ **Lemma 54.** *If $s \trianglerighteq t$ and $\mathsf{root}(t) \in \mathcal{F} \setminus \mathcal{G}$ then $s{\downarrow} \trianglerighteq t{\downarrow}$.*

Proof. Suppose $s \trianglerighteq t$ and $s \to^{n}_{\mathsf{F}(\mathcal{G})} s{\downarrow}$. By using Lemma 54 n times we derive $s{\downarrow} \trianglerighteq t'$ for some term t' with $t \to^{*}_{\mathsf{F}(\mathcal{G})} t'$. Since subterms of flattened terms are flattened, t' is flattened. Hence, $s{\downarrow} \trianglerighteq t' = t'{\downarrow} = t{\downarrow}$. ◀

Proof of Lemma 33. Suppose $s \succ_{\mathcal{G}} t$ with $s = f(s_1, \ldots, s_m)$. We have $f \in \mathcal{F} \setminus \mathcal{G}$ and $s{\downarrow} = f(s_1{\downarrow}, \ldots, s_n{\downarrow})$. By induction on the derivation of $s \succ_{\mathcal{G}} t$ we show $s{\downarrow} \succ_{\mathsf{mpo}} t{\downarrow}$.

(1) If $s \succ_{\mathcal{G}} t$ is derived by Definition 17(2) then $t = g(t_1, \ldots, t_n)$, $f \succ g$, and $s \succ_{\mathcal{G}} t_j$ for all j. Since $f \succ g$ guarantees $f \in \mathcal{F} \setminus \mathcal{G}$, the identity $s{\downarrow} = f(s_1{\downarrow}, \ldots, s_m{\downarrow})$ holds. Moreover, the induction hypothesis yields $s{\downarrow} \succ_{\mathcal{G}} t_j{\downarrow}$ for all j. Thus, the inequalities

$$s{\downarrow} = f(s_1{\downarrow}, \ldots, s_m{\downarrow}) \succ_{\mathsf{mpo}} g(t_1{\downarrow}, \ldots, t_n{\downarrow}) \succsim_{\mathsf{mpo}} t{\downarrow}$$

follow by MPO (2) and Lemma 34.

(2) If $s \succ_{\mathcal{G}} t$ is derived by Definition 17(1) then $s_i \trianglerighteq t$. We distinguish three subcases on the shape of t.

- If t is a variable then $t \in \mathcal{V}\mathsf{ar}(s_i) = \mathcal{V}\mathsf{ar}(s_i{\downarrow})$. As we have $s{\downarrow} \rhd t{\downarrow}$. Thus, the claim follows by Definition 17(1).
- If $t = g(t_1, \ldots, t_n)$ and $g \in \mathcal{F} \setminus \mathcal{G}$ then Lemma 54 yields $s_i{\downarrow} \trianglerighteq t{\downarrow}$, which leads to $s{\downarrow} \rhd t{\downarrow}$. Thus, the claim follows by Definition 17(1).
- If $t = g(t_1, \ldots, t_n)$ and $g \in \mathcal{G} \cup \mathcal{F}_{\mathsf{c}}$ then $s \rhd t_j$ for all j. So $s \succ_{\mathcal{G}} t_j$ holds for all j. Therefore, the proof for case (1) goes through. ◀

Lemma 39 claims that if s is ground and $s \trianglerighteq t$ then $s{\downarrow} \succsim_{\mathsf{mpo}} t{\downarrow}$. To facilitate its inductive proof, we show the following lemma.

▶ **Lemma 55.** *Let $C = f(s_1, \ldots, s_{i-1}, \Box, s_{i+1}, \ldots, s_n)$. If $C[t]$ is ground then $C[t]{\downarrow} \succsim_{\mathsf{mpo}} t{\downarrow}$.*

Proof. Suppose $C[t]$ is ground. We perform induction on $C[t]$ with respect to $\to^{+}_{\mathsf{F}(\mathcal{G})}$. If $t{\downarrow} = \mathsf{c}_0$ then $C[t]{\downarrow} \succsim_{\mathsf{mpo}} t{\downarrow}$ holds because $C[t]{\downarrow}$ is ground and c_0 is the minimum ground term. Suppose $t{\downarrow} \neq \mathsf{c}_0$. We proceed with analyzing f and whether s_j are flattened.

1. If $f \in \mathcal{F} \setminus \mathcal{G}$ then $C[t]{\downarrow} = f(s_1{\downarrow}, \ldots, t{\downarrow}, \ldots, s_n{\downarrow})$. So $C[t]{\downarrow} \succsim_{\mathsf{mpo}} t{\downarrow}$ is obtained by MPO (1).

2. If $f \in \mathcal{G}$ then take $D = \mathsf{c}_n(s_1, \ldots, s_{i-1}, \Box, s_{i+1}, \ldots, s_n)$. We have $C[t] \to_{\mathsf{F}(\mathcal{G})} D[t]$. Thus, we obtain $C[t]{\downarrow} = D[t]{\downarrow} \succsim_{\mathsf{mpo}} t{\downarrow}$ by the induction hypothesis.

3. Similarly, if $s_j \to^{+}_{\mathsf{F}(\mathcal{G})} s_j{\downarrow}$ for some $j \neq i$, take $D = \mathsf{c}_n(s_1{\downarrow}, \ldots, s_{i-1}{\downarrow}, \Box, s_{i+1}{\downarrow}, \ldots, s_n{\downarrow})$. The same argument applies.

4. If $f \in \mathcal{F}_{\mathsf{c}}$ and $s_j = \mathsf{c}_m(u_1, \ldots, u_n)$ for some $j \neq i$, by taking the context

$$D = \begin{cases} \mathsf{c}_n(s_1, \ldots, s_{j-1}, u_1, \ldots, u_m, s_{j+1}, \ldots, s_{i-1}, C', s_{i+1}, \ldots, s_n) & \text{if } j < i \\ \mathsf{c}_n(s_1, \ldots, s_{i-1}, C', \ldots, s_{i+1}, s_{j-1}, u_1, \ldots, u_m, s_{j+1}, \ldots, s_n) & \text{if } j > i \end{cases}$$

the claim is verified as in the last two cases.

5. Otherwise, $f = \mathsf{c}_n$ and $s_j{\downarrow} = s_j$ for all $1 \leqslant i \leqslant m$. If $n = 1$, we immediately obtain $C[t]{\downarrow} = t{\downarrow}$. So hereafter we assume $n \geqslant 2$. Furthermore we distinguish two cases, depending on $t{\downarrow}$.

- If $t{\downarrow} = \mathsf{c}_m(t_1, \ldots, t_n)$ then $m \neq 1$. As $t{\downarrow} \neq \mathsf{c}_0$, we have $m \geqslant 2$. Thus, $m + n - 1 > m$ holds. Therefore, by MPO (3) the inequality

$$C[t]{\downarrow} = \mathsf{c}_{m+n-1}(s_1, \ldots, s_{i-1}, u_1, \ldots, u_m, s_{i+1}, \ldots, s_n) \succ_{\mathsf{mpo}} t{\downarrow}$$

is derived.
- Otherwise, $C[t]{\downarrow} = \mathsf{c}_{m+n-1}(s_1, \ldots, t{\downarrow}, \ldots, s_n) \succ_{\mathsf{mpo}} t{\downarrow}$ follows by MPO (1). ◀

Proof of Lemma 39. The claim is shown by induction on s together with Lemma 55. ◀

Two-Dimensional Kripke Semantics I: Presheaves

G. A. Kavvos
University of Bristol, United Kingdom

Abstract

The study of modal logic has witnessed tremendous development following the introduction of Kripke semantics. However, recent developments in programming languages and type theory have led to a second way of studying modalities, namely through their categorical semantics. We show how the two correspond.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics; Theory of computation → Categorical semantics

Keywords and phrases modal logic, categorical semantics, Kripke semantics, duality, open maps

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.14

Related Version *Preprint*: `https://arxiv.org/abs/2405.04157`

Funding This work was supported by the UKRI Engineering and Physical Sciences Research Council grants EP/Y000242/1, EP/Y033418/1, the UKRI International Science Partnerships Fund (ISPF), and a Royal Society Research Grant.

Acknowledgements I have benefitted significantly from conversations with Dan Licata, Nachiappan Valliappan, Fabian Ruch, Amar Hadzihasanovic, Kohei Kishida, Sean Moss, Sam Staton, Daniel Gratzer, Lars Birkedal, Jonathan Sterling, Philip Saville, and Gordon Plotkin.

1 Introduction

The development of modal logic has undergone many phases [23, 15, 46, 98]. It is widely accepted that one of the most important developments was the relational semantics of Kripke [68, 69, 70] [15, §1] [46, §4.8]. Kripke semantics has proven time and again that it is intuitive and technically malleable, thereby exerting sustained influence over Computer Science.

However, over the last 30 years another way of studying modalities has evolved: looking at modal logic through the prism of the *Curry-Howard-Lambek correspondence* [72, 93, 99] yields new computational intuitions, often with surprising applications in both programming languages and formal proof. The tools of the trade here are type theory and category theory.

Up to now these two ways of looking at modalities have been discussed in isolation. The purpose of this paper is to establish a connection: I will show that the Kripke and categorical semantics of modal logic are part of a *duality*. It is well-known that dualities between Kripke and algebraic semantics exist: the *Jónsson-Tarski duality* is one of the cornerstones of classical modal logic [15, §5]. The main contribution of this paper is to show that such dualities can be elevated to the level of *proofs*. The punchline is that a *profunctor* $R : \mathcal{C}^{\mathsf{op}} \times \mathcal{C} \to \mathbf{Set}$, considered as a *proof-relevant relation* on a category $\mathcal{C}$, uniquely corresponds to a categorical model of modal logic on the category of presheaves on $\mathcal{C}$.

There are two obstacles to overcome to get to that result. The first is that we must work over an *intuitionistic* substrate: most research on types and categories is forced to do so, for unavoidable reasons [72, §8]. We must therefore first develop a duality for *intuitionistic modal logic*. However, there is no consensus on what intuitionistic modal logic is! The problem is particularly acute in the presence of $\Diamond$ [27]. I will avoid this problem by making canonical choices at each step. First, I will formulate a Kripke semantics based on *bimodules*, i.e.

relations that are canonically compatible with a poset. Then, I will show how *Kan extension* uniquely determines two adjoint modalities, $\blacklozenge$ and $\Box$, from any bimodule. The fact these arise automatically is evidence that they are the canonical choice of intuitionistic modalities.

The second obstacle stems from considering proofs. The jump from algebraic to categorical semantics involves adding an extra "dimension" of proofs. Consequently, in order to re-establish a duality, an additional dimension must be added to Kripke semantics as well. I call the result a *two-dimensional Kripke semantics*. Category theorists will find it anticlimactic: it amounts to the folklore observation that a proof-relevant Kripke semantics is essentially a semantics in a presheaf category.

Indeed, a sizeable proportion of this paper consists of folklore results that are well-known to experts. However, many of them are drawn from related but distinct areas: logic, order theory, category theory, and topos theory. As a result, it does not appear that all of them are well-known by a *single* expert. Thus, the synthesis presented here appears to be new.

The results I present in this paper show that there are deep connections between modal logic and presheaf categories. This is important, as the latter are ubiquitous in logic and related fields: presheaf models are used in fields as disparate as categorical homotopy theory [87, 24], type theory [57], concurrency [65, 21, 22], memory allocation [81, 82], synthetic guarded domain theory [14], second-order syntax and algebraic theories [34, 53, 35, 36, 37], higher-order abstract syntax [58], and so on. As a result, the connections presented here may enable synthetic reasoning *via* modalities in a variety of logical settings.

In Section 2 I recall the Kripke and algebraic semantics of intuitionistic logic, and outline the duality between Kripke semantics and certain complete Heyting algebras, the *prime algebraic lattices*. Then I extend this duality to intuitionistic modal logic in Section 3 by showing how a relation that is compatible with the intuitionistic order – a bimodule – gives rise to modalities through Kan extension. In Section 4 I add proofs to intuitionistic logic, and elevate the duality to one between "two-dimensional frames" and presheaf categories. I then repeat this exercise for intuitionistic modal logic in Section 5 by promoting bimodules to profunctors on the relational side, and adding an adjunction on the categorical side.

For general background in orders please refer to the book by Davey and Priestley [28]. Given a poset $(D, \sqsubseteq_D)$ let the *opposite* poset D^{op} be given by reversing the partial order; that is, $x \sqsubseteq_{D^{\mathsf{op}}} y$ iff $y \sqsubseteq_D x$. A *lattice* has all finite meets and joins. A *complete lattice* has arbitrary ones. A complete lattice is *infinitely distributive* just if the law $a \wedge \bigvee_i b_i = \bigvee_i a \wedge b_i$ holds. Such lattices are variously called *frames*, *locales*, or *complete Heyting algebras* [61, 76, 84].

2 Intuitionistic Logic I

There are many types of semantics for intuitionistic logic, including Kripke, Beth, topological, and algebraic semantics. Bezhanishvili and Holliday [11] argue that these form a strict hierarchy, with Kripke being the least general, and algebraic the most general. I will briefly review the elements of these extreme points of the spectrum.

The Kripke semantics of intuitionistic logic are given by *Kripke frames*, i.e. partially-ordered sets $(W, \sqsubseteq)$ [23, §2.2]. W is referred to as the set of *possible worlds*, and $\sqsubseteq$ as the *information order*. A world $w \in W$ is a "state of knowledge", and $w \sqsubseteq v$ means that moving from w to v potentially entails an increase in the amount of information.

Let $\mathsf{Up}(W)$ be the set of *upper sets* of W, i.e. the subsets $S \subseteq W$ such that $w \in S$ and $w \sqsubseteq v$ implies $v \in S$. A *Kripke model* $\mathfrak{M} = (W, \sqsubseteq, V)$ consists of a Kripke frame $(W, \sqsubseteq)$ as well as a function $V : \mathsf{Var} \to \mathsf{Up}(W)$. The *valuation* V assigns to each propositional variable $p \in \mathsf{Var}$ an upper set $V(p) \subseteq W$, which is the set of worlds in which p is true. The idea is that, once a proposition becomes true, it must remain true as information increases.

We are now able to inductively define a relation $\mathfrak{M}, w \vDash \varphi$ with the meaning that φ is true in world w of model $\mathfrak{M}$. The only interesting clause is that for implication:

$$\mathfrak{M}, w \vDash \varphi \to \psi \stackrel{\text{def}}{\equiv} \forall w \sqsubseteq v.\ \mathfrak{M}, v \vDash \varphi \text{ implies } \mathfrak{M}, v \vDash \psi$$

This definition is famously monotonic: if $\mathfrak{M}, w \vDash \varphi$ and $w \sqsubseteq v$ then $\mathfrak{M}, v \vDash \varphi$. Kripke semantics is sound and complete for intuitionistic logic [23, 15].

The algebraic semantics of intuitionistic logic consist of *Heyting algebras*. These are lattices such that every map $- \wedge x : L \to L$ has a right adjoint, i.e. for $x, y \in L$ there is an element $x \Rightarrow y \in L$ such that $c \wedge x \sqsubseteq y$ iff $c \sqsubseteq x \Rightarrow y$. Such lattices are always distributive. Assuming that one has an interpretation $[\![p]\!] \in L$ of each proposition p, each formula φ of intuitionistic logic is inductively mapped to an element $[\![\varphi]\!] \in L$ using the corresponding algebraic structure. I will not expound further on Heyting algebras; see [23, §7.3] [16, §1.1] [76, §I.8]. But I note that they are sound and complete for intuitionistic logic.

2.1 Prime algebraic lattices

Let $(W, \sqsubseteq)$ be any Kripke frame, and let $2 \stackrel{\text{def}}{=} \{0 \sqsubseteq 1\}$. Consider the poset $[W, 2]$ of monotonic functions from W to 2, ordered pointwise. This poset has a number of curious properties.

First, the monotonicity of $p : W \to 2$ means that if $p(w) = 1$ and $w \sqsubseteq v$, then $p(v) = 1$. Hence, the subset $U \stackrel{\text{def}}{=} p^{-1}(1)$ of W is an upper set. Conversely, every upper set $U \subseteq W$ gives rise to a monotonic $p_U : W \to 2$ by setting $p_U(w) = 1$ if $w \in U$, and 0 otherwise. Consequently, there is an *order bijection*

$$\mathsf{Up}(W) \cong [W, 2]$$

with the order on $\mathsf{Up}(W)$ being inclusion. I will liberally treat upper sets and elements of $[W, 2]$ as the same, but prefer the latter notation for reasons that will become clear later.

Second, given any $w \in W$, consider its *principal upper set* $\uparrow w \stackrel{\text{def}}{=} \{v \in W \mid w \sqsubseteq v\} \in [W, 2]$. This set consists of worlds with potentially more information than that found in world w. A simple argument shows that $w \sqsubseteq v$ iff $\uparrow v \subseteq \uparrow w$.[1] Thus, this gives an *order embedding*

$$\uparrow : W^{\mathsf{op}} \to [W, 2]$$

which can be shown to preserve meets and exponentials.

Third, the poset $[W, 2]$ is a *complete lattice*: arbitrary joins and meets are given pointwise. Viewing the elements of $[W, 2]$ as upper sets, these joins and meets correspond to arbitrary unions and intersections of upper sets, which are also upper. Moreover, this lattice satisfies the infinite distributive law, so it is a *complete Heyting algebra* – synonymously a *frame* or *locale* [61, 84]. Given two upper sets $X, Y \subseteq W$ their exponential is given by [29, §1.9]

$$X \Rightarrow Y \stackrel{\text{def}}{=} \{w \in W \mid \forall w \sqsubseteq v.\ v \in X \text{ implies } v \in Y\}$$

Fourth, the principal upper sets $\uparrow w$ are special, in that they are *prime*.[2] An element d of a complete lattice L is *prime* just if

$$d \sqsubseteq \bigsqcup X \text{ implies } \exists x \in X.\ d \sqsubseteq x$$

[1] This is an order-theoretic consequence of the Yoneda lemma.

[2] Such elements are variously called *completely join-irreducible* [86], *supercompact* [8] [84, §VII.8], *completely (join-)prime* [100], or simply *join-prime* [41, §1.3].

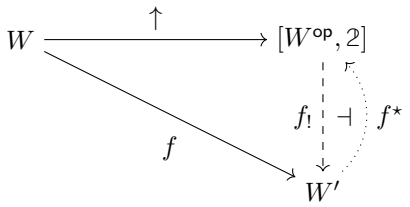
This says that d contains a tiny, indivisible fragment of information: as soon as it approximates a supremum, it must approximate something in the set that is being upper-bounded. The prime elements of $[W, 2]$ are exactly the principal upper sets $\uparrow w$ for some $w \in W$.

Fifth, the complete lattice $[W, 2]$ is *prime algebraic*. This means that all its elements can be reconstructed by "multiplying" or "sticking together" prime elements. In symbols, a complete lattice L is prime algebraic whenever for every element $d \in L$ we have

$$d = \bigsqcup \{p \in L \mid p \sqsubseteq d, p \text{ prime } \}$$

Such lattices are variously called *completely distributive, algebraic lattices* [28, §10.29] or *superalgebraic lattices* [84, §VII.8]. In fact, it can be shown that any such lattice is essentially of the form $[W, 2]$, i.e. a lattice of upper sets; this was shown by Raney in the 1950s [86], and independently by Nielsen, Plotkin and Winskel in the 1980s [80]. See the paper by Winskel for the use of prime algebraic lattices in semantics [100].

Finally, the fact every element can be reconstructed as a supremum of primes means that it is possible to canonically extend any monotonic $f : W \to W'$ to a monotonic $[W^{\mathsf{op}}, 2] \to W'$, as long as W' is a complete lattice. Diagrammatically, in the situation



(1)

there exists a unique $f_!$ which preserves all joins and satisfies $f_!(\uparrow w) = f(w)$. It is given by

$$f_!(S) \stackrel{\text{def}}{=} \bigsqcup \{f(w) \mid w \in S\}$$

$f_!$ is called the *(left) Kan extension* of f along $\uparrow$. As $f_!$ preserves all joins and $[W, 2]$ is complete it has a right adjoint $f^\star$, by the adjoint functor theorem [28, §7.34] [61, §I.4.2]. For any complete lattice W' this situation amounts to a bijection

$$\mathrm{Hom}_{\mathbf{Pos}}(W, W') \cong \mathrm{Hom}_{\mathbf{CSLatt}}([W^{\mathsf{op}}, 2], W')$$

where **CSLatt** is the category of complete lattices and join-preserving maps.

Suppose then that we have a Kripke model $(W, \sqsubseteq, V)$. The construction given above induces a Heyting algebra $[W, 2]$. Defining $[\![p]\!] \stackrel{\text{def}}{=} V(p)$ we obtain an algebraic model of intuitionistic logic, which interprets every formula φ as an upper set $[\![\varphi]\!] \in [W, 2]$. This is the upper set of worlds in which a formula is true [23, Theorem 7.20]:

▶ **Theorem 1.** *$w \vDash \varphi$ if and only if $w \in [\![\varphi]\!]$.*

Thus, every Kripke semantics corresponds to a prime algebraic lattice.

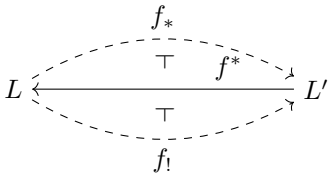
▶ Remark 2. This shows that a Kripke semantics is a particular kind of algebraic semantics. Thus, we can deduce the completeness of the latter from the completeness of the former: if a formula is valid in all Heyting algebras, it must be valid in all prime algebraic lattices, and hence valid in all Kripke semantics. If the Kripke semantics is complete, then the formula must be provable. Therefore, the algebraic semantics is then complete as well.

The opposite direction – viz. proving the completeness of Kripke semantics from completeness of the algebraic semantics – cannot be shown constructively. The reason is that it requires the construction of *prime filters*, which is a weak form of choice. I will investigate the details of this mismatch in a sequel paper.

2.2 Morphisms

The simplest kind of morphism between Kripke frames is a *monotonic* map $f : W \to W'$. Frames and monotonic maps form the category **Pos** of posets. Given a monotonic $f : W \to W'$ we can define a monotonic $f^* : [W', 2] \to [W, 2]$ by taking $p : W' \to 2$ to $p \circ f : W \to 2$. Viewing the elements of $[W', 2]$ as upper sets, f^* maps the upper set $S \subseteq W'$ to the set $\{v \in W' \mid f(v) \in S\} \subseteq W$, which is upper by the monotonicity of f. f^* preserves arbitrary joins and meets. It is thus the morphism part of a functor $[-, 2] : \mathbf{Pos}^{\mathsf{op}} \longrightarrow \mathbf{PrAlgLatt}$ to the category **PrAlgLatt** of prime algebraic lattices and complete lattice homomorphisms.

Moreover, the functor $[-, 2]$ is an equivalence! By the adjoint functor theorem any complete lattice homomorphism $f^* : L' \to L$ has a left and right adjoint:



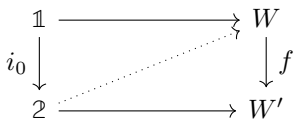
(2)

Given a prime algebraic lattice L, let $\mathsf{Prm}(L) \subseteq L$ be the sub-poset of prime elements. It can be shown that the left adjoint $f_!$ maps primes to primes [41, Lemma 1.23]. We can thus restrict it to a function $\mathsf{Prm}(L) \to \mathsf{Prm}(L')$. This defines a functor $\mathsf{Prm}(-) : \mathbf{PrAlgLatt} \longrightarrow \mathbf{Pos}^{\mathsf{op}}$ with the property that $\mathsf{Prm}([W, 2]) \cong W$. All in all, this amounts to a *duality*

$$\mathbf{Pos}^{\mathsf{op}} \simeq \mathbf{PrAlgLatt} \tag{3}$$

However, monotonic maps are not particularly well-behaved from the perspective of logic, as they do not preserve nor reflect "local" truth. This is the privilege of *open maps*.

▶ **Definition 3.** *Let $i_0 : \mathbb{1} \to 2$ map the unique point of $\mathbb{1} \stackrel{def}{=} \{*\}$ to $0 \in 2$. A monotonic map $f : W \to W'$ of Kripke frames is* open *just when it has the right lifting property with respect to $i_0 : \mathbb{1} \to 2$, i.e. when every commuting diagram of the form*



*in **Pos** has a diagonal filler (dashed) that makes it commute.*

In other words, f is open if whenever $f(w) \sqsubseteq v'$ there exists a $w' \in W$ with $w \sqsubseteq w'$ and $f(w') = v'$.[3] Open maps send upper sets to upper sets [23, Prop. 2.13]. Thus

▶ **Lemma 4.** *Let $\mathfrak{M} = (W, \sqsubseteq, V)$ and $\mathfrak{N} = (W', \sqsubseteq, V')$ be Kripke models, and $f : W \to W'$ be open. Suppose $V = f^{-1} \circ V'$, i.e. $w \in V(p)$ iff $f(w) \in V'(p)$. Then $\mathfrak{M}, w \vDash \varphi$ iff $\mathfrak{N}, f(w) \vDash \varphi$.*

Write $W \vDash \varphi$ to mean that $(W, \sqsubseteq, V), w \vDash \varphi$ for any valuation V and $w \in W$. Then

▶ **Lemma 5.** *If $f : W \to W'$ is open and surjective, then $W \vDash \varphi$ implies $W' \vDash \varphi$.*

[3] Such morphisms are often called *p-morphisms* [23, §2.3] or *bounded morphisms* [15, §2.1]. According to Goldblatt [46] open maps were introduced by de Jongh and Troelstra [29] in intuitionistic logic, and by Segerberg [91] in modal logic. More rarely they are called *functional simulations*, and led us to bisimulations [90, §3.2]. The name is chosen because such maps are open with respect to the *Alexandrov topology* on a poset, whose open sets are the upper sets [61, §1.8].

Recall now the induced map $f^* : [W', 2] \to [W, 2]$ for a monotonic $f : W \to W'$. The following lemma allows us to characterise the openness and surjectivity of f in terms of f^*.

▶ **Lemma 6.**

1. *$f : W \to W'$ is open iff $f^* : [W', 2] \to [W, 2]$ preserves exponentials.*
2. *$f : W \to W'$ is surjective iff $f^* : [W', 2] \to [W, 2]$ is injective.*

Consequently, the duality (3) may be restricted to two wide subcategories:

$$\mathbf{Pos}^{\mathsf{op}}_{\mathrm{open}} \simeq \mathbf{PrAlgLatt}_{\Rightarrow} \qquad \mathbf{Pos}^{\mathsf{op}}_{\mathrm{open,surj}} \simeq \mathbf{PrAlgLatt}_{\Rightarrow,\mathrm{inj}} \tag{4}$$

The morphisms of the categories to the left of $\simeq$ are open (resp. and surjective) maps, and the morphisms of the categories to its right are *complete Heyting homomorphisms*, i.e. complete lattice homomorphisms that preserve exponentials (resp. and are injective).

Finally, let us consider the classical case – as a sanity-check. This amounts to restricting **Pos** to its subcategory of discrete orders, i.e. **Set**. In this case every map is open. The corresponding restriction on the other side is to the category **CABA** of *complete atomic Boolean algebras*, yielding the usual *Tarski duality* $\mathbf{Set}^{\mathsf{op}} \simeq \mathbf{CABA}$ [67].

2.3 Related work

The origins of the construction of a Heyting algebra from a Kripke frame seems to be lost in the mists of time. The earliest occurrence I have located is in the book by Fitting [39, §1.6], where it is attributed to an exercise in the book by Beth [10].

The duality (3) appears to be folklore – folklore enough to be included as an exercise in new textbooks [41, Ex. 1.3.10]; see also Erné [31]. However, I have not been able to find any mention of the dualities of (4) in the literature.

Both the dualities (3) and (4) involve just prime algebraic lattices, which is a far cry from encompassing all Heyting algebras. It is possible to do so, by enlarging the category **Pos** to a class of ordered topological spaces called *descriptive frames* [23, §8.4]. The resulting duality is called *Esakia duality* [32] [41, §4.6] [12, §2.3].

A survey on dualities for classical modal logic is given by Kishida [67].

3 Modal Logic I

The task now is to extend the results of Section 2 to *intuitionistic modal logic*.

There is disagreement on what a minimal intuitionistic modal logic is. This arises no matter the methodology we choose – be it relational, algebraic, or proof-theoretic. The situation becomes even more complex if we include a diamond modality ($\Diamond$): see Das and Marin [27] and Wolter and Zakharyaschev [103] for a discussion.

In this paper I will adopt the *intuitionistic propositional logic with Galois connections* of Dzik, Järvinen, and Kondo [30], for reasons that will become clear in a moment. This extends intuitionistic logic with modalities $\blacklozenge$ and $\Box$, and the two inference rules

$$\frac{\blacklozenge\varphi \to \psi}{\varphi \to \Box\psi} \qquad \text{and} \qquad \frac{\varphi \to \Box\psi}{\blacklozenge\varphi \to \psi}$$

These rules correspond to a *Galois connection* [28, §7.23], i.e. an adjunction $\blacklozenge \dashv \Box$ between posets. They imply the derivability of the following rules, amongst others [30, Prop. 2.1].

$$\frac{\varphi \to \psi}{\Box\varphi \to \Box\psi} \qquad \frac{\varphi}{\Box\varphi} \qquad \frac{}{\Box\top} \qquad \frac{\blacklozenge\bot}{\bot} \qquad \frac{\varphi \to \psi}{\blacklozenge\varphi \to \blacklozenge\psi} \qquad \frac{}{\blacklozenge(\varphi \vee \psi) \leftrightarrow \blacklozenge\varphi \vee \blacklozenge\psi}$$

$$\frac{}{\Box(\varphi \wedge \psi) \leftrightarrow \Box\varphi \wedge \Box\psi}$$

The notation of the "black diamond" modality $\blacklozenge$ may appear unusual. However, I will argue that this logic is, in a way, the canonical intuitionistic modal logic.

The Kripke semantics of classical modal logic is given by a *modal frame* (W, R), which consists of a set W and an arbitrary *accessibility relation* $R \subseteq W \times W$ [15, §1]. If the same set of worlds W is already part of an intuitionistic Kripke frame $(W, \sqsubseteq)$ we must take care to ensure that $\sqsubseteq$ and R are *compatible*. There are many compatibility conditions that one can consider [85] [92, §3.3]. However, I will take a hint from the category theory literature, and seek a canonical definition of what it means for a relation to be compatible with a poset.

Recall that relations can be presented as functions $R : W \times W \to 2$ which map a pair of worlds (w, v) to 1 iff $w \, R \, v$. I will ask that R is such function, but with a twist:

▶ **Definition 7.** *A* bimodule $R : W_1 \nrightarrow W_2$ *is a monotonic map* $R : W_1^{\mathsf{op}} \times W_2 \to 2$.

Thus, a relation $R \subseteq W_1 \times W_2$ is a bimodule just if $w' \sqsubseteq w \, R \, v \sqsubseteq v'$ implies $w' \, R \, v'$. This means that R can absorb changes in information on either side: contravariantly on the first component, and covariantly on the second. This is a standard, minimal way to define what it means to be "a relation in **Pos**". It is strongly reminiscent of bimodules in abstract algebra.

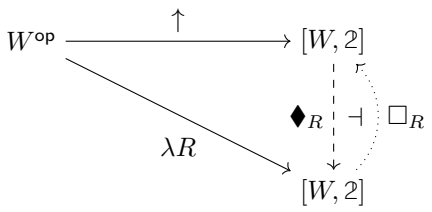
We can then define a *modal Kripke frame* $(W, \sqsubseteq, R)$ to be a Kripke frame $(W, \sqsubseteq)$ equipped with a bimodule $R : W \nrightarrow W$. A *modal Kripke model* $\mathfrak{M} = (W, \sqsubseteq, R, V)$ adds to this a function $V : \mathsf{Var} \to \mathsf{Up}(W)$. We extend $\mathfrak{M}, w \vDash \varphi$ to modal formulae:

$$\mathfrak{M}, w \vDash \blacklozenge\varphi \overset{\text{def}}{\equiv} \exists v.\ v \, R \, w \text{ and } \mathfrak{M}, v \vDash \varphi$$
$$\mathfrak{M}, w \vDash \Box\varphi \overset{\text{def}}{\equiv} \forall v.\ w \, R \, v \text{ implies } \mathfrak{M}, v \vDash \varphi$$

There are a number of things to note about this definition. First, there is a deep duality between the clauses: not only do we exchange $\forall$ for $\exists$, but we also flip the variance of R. As a result, $\blacklozenge$ uses the relation in the *opposite variance* to the more traditional $\Diamond$ modality (hence the change in notation). Second, the clause for the $\Box$ modality is the traditional one; some streams of work on intuitionistic modal logic adopt a slightly different one [85, 92], which is equivalent to this in the presence of the bimodule condition. Finally, this definition is monotonic: the bimodule conditions on R suffice to show that if $\mathfrak{M}, w \vDash \varphi$ and $w \sqsubseteq v$ then $\mathfrak{M}, v \vDash \varphi$. Dzik et al. [30, §5] prove that this semantics is sound and complete.

The algebraic semantics of this logic is given by a Heyting algebra H equipped with two monotonic maps $\blacklozenge, \Box : H \to H$ which form an adjunction $\blacklozenge \dashv \Box$, i.e. a Galois connection. Dzik et al. [30, §4] prove that this semantics is also sound and complete.

We are now in a position to relate the Kripke and algebraic semantics of this intuitionistic modal logic. Let $(W, \sqsubseteq, R)$ be a modal Kripke frame, and consider the map $\lambda R : W^{\mathsf{op}} \to [W, 2]$ obtained by cartesian closure of **Pos**. This map takes $w \in W$ to the upper set $\{v \in W \mid w \, R \, v\}$ of worlds accessible from w. Putting λR in (1) we obtain through Kan extension the diagram



(5)

where we write $\blacklozenge_R$ for $\lambda R_!$ and $\Box_R$ for $\lambda R^\star$. It can be shown that these maps are given by

$$\blacklozenge_R(S) \stackrel{\text{def}}{=} \{w \in W \mid \exists v.\ v\, R\, w \text{ and } v \in S\}$$
$$\Box_R(S) \stackrel{\text{def}}{=} \{w \in W \mid \forall v.\ w\, R\, v \text{ implies } v \in S\}$$

Thus, any bimodule R defines an adjunction $\blacklozenge_R \dashv \Box_R$ on $[W, 2]$. Correspondingly, any adjunction $\blacklozenge \dashv \Box$ on $[W, 2]$ yields a monotonic map $\blacklozenge \circ \uparrow(-) : W^{\mathsf{op}} \to [W, 2]$, which uniquely corresponds to a bimodule $W^{\mathsf{op}} \times W \to 2$ by the cartesian closure of **Pos**.

Thus, starting from a bimodule, i.e. a relation that is compatible with the information order, we have canonically obtained a model of intuitionistic modal logic on $[W, 2]$ through Kan extension: $[W, 2]$ is a complete Heyting algebra, and we define $[\![\blacklozenge\varphi]\!] = \blacklozenge_R[\![\varphi]\!]$ and $[\![\Box\varphi]\!] = \Box_R[\![\varphi]\!]$. We immediately obtain a modal analogue to Theorem 1:

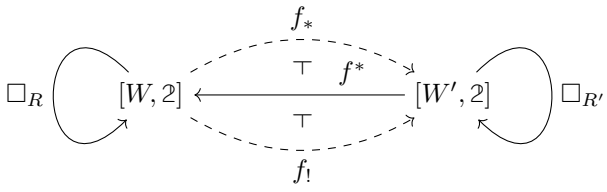
▶ **Theorem 8.** *For any modal formula φ, $w \vDash \varphi$ if and only if $w \in [\![\varphi]\!]$.*

3.1 Morphisms

We define a category **Bimod** with bimodules $R : W_1 \nrightarrow W_2$ as objects. A *bimodule morphism* from $R : W_1 \nrightarrow W_2$ to $R' : W_1' \nrightarrow W_2'$ is a pair (f, g) of monotonic maps $f : W_1 \to W_1'$ and $g : W_2 \to W_2'$ such that $R(w, v) \sqsubseteq R'(f(w), g(v))$. Stated in terms of relations, it must be that $w\, R\, v$ implies $f(w)\, R'\, g(v)$.

We define the subcategory **EBimod** to consist of (endo)bimodules $R : W \nrightarrow W$ and pairs of maps (f, f). Thus, objects are bimodules on a single poset W, and morphisms are monotonic maps $f : W \to W'$ that preserve the relation, i.e. $w\, R\, v$ implies $f(w)\, R\, f(v)$. In other words, the objects of **EBimod** are modal Kripke frames, and the morphisms are monotonic, relation-preserving maps.

Recall the adjunctions and modalities induced by a monotonic $f : W \to W'$:



(6)

▶ **Lemma 9.** *$f : W \to W'$ is a morphism of bimodules $f : R \to R'$ iff $f^* \Box_{R'} \subseteq \Box_R f^*$.*

This constitutes a duality

$$\mathbf{EBimod}^{\mathsf{op}} \simeq \mathbf{PrAlgLattO} \tag{7}$$

where **PrAlgLattO** is the category with objects $(L, \Box_L)$, where L is a prime algebraic lattice and $\Box_L : L \to L$ is an operator that preserves all meets. By the adjoint functor theorem, such operators always have a left adjoint $\blacklozenge_L : L \to L$. Thus, this category contains algebraic

models of intuitionistic modal logic (but not all of them). By the preceding section each such adjunction corresponds uniquely to a bimodule. The morphisms of **PrAlgLattO** are complete lattice homomorphisms $h : L \to L'$ such that $h\Box_L \sqsubseteq \Box_{L'}h$. By the preceding lemma they correspond precisely to morphisms of bimodules.

However, as with monotone maps, morphisms of bimodules do not preserve local truth; for that we need a notion of *modally open* maps.

▶ **Definition 10.** *Let $(W, \sqsubseteq, R)$ and $(W', \sqsubseteq, R')$ be modal Kripke frames. A bimodule morphism $f : R \to R'$ is* modally open *just if whenever $f(w)\ R'\ v$ then there exists a $w' \in W$ with $w\ R\ w'$ and $f(w') \sqsubseteq v$.*

This is similar to Definition 3, but ever so slightly weaker: instead of requiring $f(w') = v'$, it requires that the information in $f(w')$ can be increased to v'. Like Definition 3, it can also be written homotopy-theoretically, but that requires some ideas from double categories that are beyond the scope of this paper. We have the analogous result about preservation of truth:

▶ **Lemma 11.** *Let $\mathfrak{M} = (W, \sqsubseteq, R, V)$ and $\mathfrak{N} = (W', \sqsubseteq, R', V')$ be modal Kripke models, $f : W \to W'$ be open and modally open, and $V = f^{-1} \circ V'$. Then $\mathfrak{M}, w \vDash \varphi$ iff $\mathfrak{N}, f(w) \vDash \varphi$.*

▶ **Lemma 12.** *Let $f : W \to W'$ be open, modally open, and surjective. If $W \vDash \varphi$ then $W' \vDash \varphi$.*

The following result relates the modal openness of f to f^*.

▶ **Lemma 13.** *$f : R \to R'$ is modally open iff $\Box_R f^* = f^*\Box_{R'}$ iff $f_!\blacklozenge_R = \blacklozenge_{R'}f_!$.*

Thus, the duality (7) may be restricted to dualities between wide subcategories:

$$\mathbf{EBimod}^{\mathrm{op}}_{\mathrm{moo}} \simeq \mathbf{PrAlgLattO}_{\Rightarrow o} \qquad \mathbf{EBimod}^{\mathrm{op}}_{\mathrm{moo,\ surj}} \simeq \mathbf{PrAlgLattO}_{\Rightarrow o,\mathrm{inj}} \tag{8}$$

The morphisms to the left of $\simeq$ are open and modally open (resp. and surjective); and the to the right of it preserve exponentials and commute with operators (resp. and are injective).

Let us consider the restriction of this duality to the classical setting – as a sanity check. A bimodule on a discrete poset is just a relation on a set. The corresponding restriction on the right is to CABAs with operators, and complete homomorphisms which commute with operators. We thus obtain the *Thomason duality* $\mathbf{MFrm}^{\mathrm{op}}_{\mathrm{open}} \simeq \mathbf{CABAO}$ between Kripke frames and modally open maps on the left, and CABAs with operators to the right [96, 67].

3.2 Related work

Many works have presented a Kripke semantics for intuitionistic modal logic. All such semantics assume two accessibility relations: a preorder for the intuitionistic dimension, and a second relation for the modal dimension. What varies is their *compatibility conditions*.

The first work to present such a semantics appears to be that of Fischer Servi [38]. One of the required compatibility conditions is $(\sqsubseteq) \circ R \subseteq R \circ (\sqsubseteq)$. This is weaker than having a bimodule, but sufficient to prove soundness.

The first work to recognise the importance of bimodules was Sotirov's 1979 thesis. His results are summarised in a conference abstract [94, §4]: they include the completeness of a minimal intuitionistic modal logic with a $\Box$, the K axiom, and the necessitation rule. Božić and Došen [19] repeat the study for the same logic, but for a semantics based on the Fischer Servi compatibility conditions. However, they note that their completeness proof actually constructs half a bimodule (a "condensed" relation). They also point out that bimodules, which they call "strictly condensed" relations, are sound and complete for their logic. Wolter and Zakharyaschev [101, §2] argue that bimodule and Fischer Servi semantics are equi-expressive.

Plotkin and Stirling [85] attempt to systematise the Kripke semantics of intuitionistic modal logic. Their frame conditions allow "transporting a modal relation upwards" along any potential increases of information on either side. This paper and all its descendants – notably the thesis of Simpson [92, §3.3] – adopt a different satisfaction clause for $\Box$ which uses both $\sqsubseteq$ and R. In the presence of the bimodule conditions this satisfaction clause is equivalent to the classical one, which I use here.

The bimodule condition and the *complex algebra* construction (or fragments thereof) have made scattered appearances in the literature: in the early work of Sotirov [94] and Božić and Došen [19]; in the work of Wolter and Zakharyaschev [102, 101, 103], Hasimoto [54, §4], and Orłowska and Rewitzky [83]; and of course in Dzik et al. [30, §7]. With the exception of the last one, none of these references discuss the $\blacklozenge$ modality. Moreover, in none of these references are the categorical aspects of this construction discussed.

As mentioned before, dualities between frames and algebras have played a significant role in modal logic. Thomason [96] and Goldblatt [45] also considered morphisms of frames, respectively obtaining *Thomason duality* and (categorical) *Jónsson-Tarski duality* between descriptive frames and Boolean Algebras with Operators (BAOs) [46, §6.5]. Kishida [67] surveys a number of dualities for classical modal logic.

The duality (7) is stated by Gehrke [40, Thm. 2.5] who attributes it to Jónnson [64], even though no such theorem appears in that paper.

The dualities of (8) are the direct intuitionistic analogues to that of Thomason. I have not been able to find them anywhere in the literature.

According to the extensive survey of Menni and Smith [78], the idea that the commonly-used modalities $\Box$ and $\Diamond$ are often part of adjunctions $\blacklozenge \dashv \Box$ and $\Diamond \dashv \blacksquare$ is implicitly present throughout the development of modal logic. However, these were not made explicit in a logic until the 2010s, when they appeared in the work of Dzik et al. [30] and Sadrzadeh and Dyckhoff [89]. The same perspective plays a central rôle in the exposition of Kishida [67].

The $\blacklozenge$ modality has appeared before in *tense logics* as a "past" modality [33, 47].

4 Intuitionistic Logic II

In the rest of this paper we will *categorify* [7] the notion of Kripke semantics. The main idea is to replace posets by categories, so that the order $w \sqsubseteq v$ is replaced by a morphism $w \to v$. As there might be multiple morphisms $w \to v$, this allows the recording of not just the fact v may signify more information than w, but also the *manner* in which it does so. The reflexivity and transitivity of the poset are then replaced by the identity and composition laws of the category. This adds a dimension of *proof-relevance* to Kripke semantics.

A corresponding change in our algebraic viewpoint will be that of replacing the set 2 of truth values with the category **Set**. This is a classic Lawverean move [73]. Notice that this is lopsided, as is usual in intuitionistic logic: while the falsity 0 is only represented by one value, viz. the empty set, the truth 1 can be represented by any non-empty set X. The elements of X can be thought of as a *proofs* of a true statement.

Let us then trade the frame $(W, \sqsubseteq)$ for an arbitrary category $\mathcal{C}$. It remains to define what it means to have a *proof* that the formula φ holds at a world $w \in \mathcal{C}$. We denote the set of all such proofs by $[\![\varphi]\!]_w$. Assuming we are given a set $[\![p]\!]_w$ for each proposition p and world w, here is a first attempt:

$$[\![\bot]\!]_w \stackrel{\text{def}}{=} \emptyset \qquad [\![\top]\!]_w \stackrel{\text{def}}{=} \{*\} \qquad [\![\varphi \wedge \psi]\!]_w \stackrel{\text{def}}{=} [\![\varphi]\!]_w \times [\![\psi]\!]_w \qquad [\![\varphi \vee \psi]\!]_w \stackrel{\text{def}}{=} [\![\varphi]\!]_w + [\![\psi]\!]_w$$

$$[\![\varphi \to \psi]\!]_w \stackrel{\text{def}}{=} (v : \mathcal{C}) \to \mathrm{Hom}_{\mathcal{C}}(w, v) \to [\![\varphi]\!]_v \to [\![\psi]\!]_v$$

where for a family of sets $(B_a)_{a\in A}$ we let

$$(a:A)\to B_a \stackrel{\text{def}}{=} \left\{ f : A \to \bigcup_{a\in A} B_a \;\middle|\; \forall a \in A. f(a) \in B_a \right\}$$

This closely follows the usual Kripke semantics, but adds proofs. For example, a proof in $[\![\varphi_1 \wedge \varphi_2]\!]_w$ is a pair (x, y) of a proof $x \in [\![\varphi_1]\!]_w$ and a proof $y \in [\![\varphi_2]\!]_w$. Similarly, a proof $F \in [\![\varphi \to \psi]\!]_w$ is a function which maps a proof of "increase in information" $f : w \to v$ to a function $F(v)(f) : [\![\varphi]\!]_v \to [\![\psi]\!]_v$. In turn, this function maps proofs in $[\![\varphi]\!]_v$ to proofs in $[\![\psi]\!]_v$.

To show that this definition is monotonic we have to demonstrate it on proofs: given a proof $x \in [\![\varphi]\!]_w$ and a morphism $f : w \to v$ we have to define a proof $f \cdot x \in [\![\varphi]\!]_v$. Assuming that we are given this operation for propositions, we can extend it by induction; e.g.

$$\begin{array}{lcll} f \cdot (x, y) & \stackrel{\text{def}}{=} & (f \cdot x, f \cdot y) & \in [\![\varphi \wedge \psi]\!]_v \\ f \cdot F & \stackrel{\text{def}}{=} & (z : \mathcal{C}) \mapsto (g : \mathrm{Hom}_{\mathcal{C}}(v, z)) \mapsto (x : [\![\varphi]\!]_z) \mapsto F(z)(g \circ f)(x) & \in [\![\varphi \to \psi]\!]_v \end{array}$$

Moreover, this definition is compatible with $\mathcal{C}$, in the sense that $g \cdot (f \cdot x) = (g \circ f) \cdot x$ and $\mathsf{id}_w \cdot x = x$. We thus obtain a (covariant) *presheaf* $[\![\varphi]\!] : \mathcal{C} \longrightarrow \mathbf{Set}$ for each formula φ.

It is well-known that the proofs of intuitionistic logic form a *bicartesian closed category* (biCCC), i.e. a category with finite (co)products and exponentials [71]. A biCCC can be seen as a categorification of a Heyting algebra: formulae are objects of the category, and proofs are morphisms. We will not expound on this further; see [72, 26, 4].

It should therefore be the case that the semantics described above form a biCCC. Indeed, it is a well-known fact of topos theory that the *category of presheaves* $[\mathcal{C}, \mathbf{Set}]$ is a biCCC. In fact, the construction of exponentials [76, §I.6] reveals that our definition above is deficient: we should restrict $[\![\varphi \to \psi]\!]_w$ to contain only those functions F that satisfy a *naturality condition*, i.e. those which for any $f : w \to v_1$, $g : v_1 \to v_2$, and $x \in [\![\varphi]\!]_{v_1}$ satisfy

$$g \cdot F(v_1)(f)(x) = F(v_2)(g \circ f)(g \cdot x)$$

From this point onwards I will identify two-dimensional Kripke semantics with categorical semantics in a category of presheaves $[\mathcal{C}, \mathbf{Set}]$.

4.1 Presheaf categories

The category $[\mathcal{C}, \mathbf{Set}]$ of covariant presheaves is eerily similar to prime algebraic lattices. In a sense they are just the same; but, having traded $\mathbb{2}$ for **Set**, they have become proof-relevant.

First, letting $P \in [\mathcal{C}, \mathbf{Set}]$, an element $x \in P(w)$ is a proof that P holds at a "world" $w \in \mathcal{C}$. A morphism $f : w \to v$ of $\mathcal{C}$ then leads to a proof $f \cdot x \stackrel{\text{def}}{=} P(f)(x) \in P(v)$ that P holds at v. Thus, the presheaf P is very much like an upper set.

Second, the *representable presheaves* $\mathbf{y}(w) \stackrel{\text{def}}{=} \mathrm{Hom}_{\mathcal{C}}(w, -) : \mathcal{C} \to \mathbf{Set}$ are the proof-relevant analogues of the principal upper set. By the Yoneda lemma they constitute an *embedding*

$$\mathbf{y} : \mathcal{C}^{\mathsf{op}} \longrightarrow [\mathcal{C}, \mathbf{Set}]$$

which moreover preserves limits and exponentials [4].

Third, the category $[\mathcal{C}, \mathbf{Set}]$ is both complete and cocomplete, with limits and colimits computed pointwise [76, §I]. It is also "distributive" in an appropriate sense [3, §3.3], which makes it into a *Grothendieck topos*. It is thus a cartesian closed category, with exponential

$$(P \Rightarrow Q)(w) \stackrel{\text{def}}{=} \mathrm{Hom}(P \times \mathbf{y}(w), Q)$$

which is essentially the two-dimensional semantics of implication I gave above.

Fourth, the representables $\mathbf{y}(w)$ are special, in that they are *tiny* [104].

▶ **Definition 14.** *An object $d \in \mathcal{D}$ is* tiny *just if* $\mathrm{Hom}(d, -) : \mathcal{D} \to \mathbf{Set}$ *preserves colimits.*[4]

Tininess is a proof-relevant version of primality: it implies that for any $f : w \to \varinjlim_i v_i$ there exists an i such that f is equal to the composition of a morphism $w \to v_i$ with the injection $v_i \to \varinjlim_i v_i$. By the Yoneda lemma it follows that all representables $\mathbf{y}(w)$ are tiny, as they satisfy the above definition for $\mathcal{D} \stackrel{\text{def}}{=} [\mathcal{C}, \mathbf{Set}]$ and $d \stackrel{\text{def}}{=} \mathbf{y}(w)$.

Fifth, the so-called *co-Yoneda lemma* [75, §III.7] shows that every $P \in [\mathcal{C}, \mathbf{Set}]$ is a colimit of representables. This means that it can be reconstructed by sticking together tiny elements:

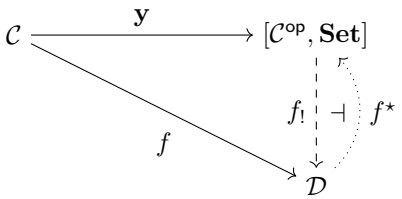
$$P \cong \varinjlim_{(w,x)\in \mathsf{el}\, P} \mathbf{y}(w)$$

Like with prime algebraic lattices, there is a converse to this result: every category which is generated by sticking together tiny elements is in fact a presheaf category:

▶ **Theorem 15** (Bunge [20])**.** *A category which is cocomplete and strongly generated by a small set of tiny objects is equivalent to* $[\mathcal{C}, \mathbf{Set}]$ *for some small category* $\mathcal{C}$*.*

A textbook presentation of this result can be found in the book by Kelly [66, §5.5].

Finally, the fact every element can be reconstructed as a colimit of representables means that it is possible to uniquely extend any functor $f : \mathcal{C} \longrightarrow \mathcal{D}$ to a cocontinuous functor $[\mathcal{C}^{\mathsf{op}}, 2] \longrightarrow \mathcal{D}$, as long as $\mathcal{D}$ is cocomplete. Diagrammatically, in the situation



(9)

there exists an essentially unique cocontinuous $f_!$ with $f_!(\mathbf{y}(w)) = f(w)$. It is given by

$$f_!\Big(\varinjlim_{(w,x)\in \mathsf{el}\, P} \mathbf{y}(w)\Big) \stackrel{\text{def}}{=} \varinjlim_{(w,x)\in \mathsf{el}\, P} f(w)$$

$f_!$ is called the *left Kan extension* of f along $\mathbf{y}$. It has a right adjoint $f^\star$ which is explicitly given by $f^\star(d) \stackrel{\text{def}}{=} \mathrm{Hom}(f(-), d)$. This amounts to an isomorphism

$$\mathrm{Hom}_{\mathbf{Cat}}(\mathcal{C}, \mathcal{D}) \cong \mathrm{Hom}_{\mathbf{Cocont}}([\mathcal{C}^{\mathsf{op}}, \mathbf{Set}], \mathcal{D})$$

where **Cat** is the category of categories, and **Cocont** is the category of cocomplete categories and cocontinuous functors: see [4, Prop. 9.16] [88, Cor. 6.2.6, Rem. 6.5.9] and [75, § X.3, Cor. 2] [66, Th. 4.51].

All in all, presheaf categories are the categorification of prime algebraic lattices.

4.2 Cauchy-complete and spacelike categories

Replacing posets with categories does not come for free: the extra dimension of morphisms leads to situations that have no analogues in poset. Some of these are problematic when thinking of $\mathcal{C}$ as a two-dimensional Kripke frame. Perhaps the most bizarre is the presence of *idempotents*, i.e. morphisms $e : w \to w$ with the property that $e \circ e = e$. Such morphisms represent a non-trivial increase in information which confusingly leaves us in the same world.

[4] In the literature this property is often referred to as *external tininess* (cf. internal tininess).

Table 1 Categorification of Kripke semantics.

poset	category
monotonic map	functor
upper sets	presheaves
principal upper set	representable presheaf
prime element	tiny object
prime algebraic lattice	presheaf category
bimodule	profunctor

The presence of idempotents causes issues. For example, recall that, in prime algebraic lattices, primes and principal upper sets coincide. The astute reader will have noticed we did *not* claim the analogous result in presheaf categories: tiny objects are not necessarily representable in $[\mathcal{C}, \mathbf{Set}]$. For that, we need $\mathcal{C}$ to be *Cauchy-complete* [18, 17].

▶ **Definition 16.** *A category is* Cauchy-complete *just if every idempotent splits, i.e. if every idempotent is equal to $s \circ r$ for a section-retraction pair s and r.*

Note that every complete category is Cauchy-complete, including **Set** and $[\mathcal{C}, \mathbf{Set}]$.

This leads us to another troublesome situation, namely that of having section-retraction pairs, i.e. $s : w \to v$ and $r : v \to w$ with $r \circ s = \mathsf{id}_w$. In this case w and v contain no more information than each other, but are not isomorphic. We may ask that this does not arise.

▶ **Definition 17.** *A category satisfies the* Hemelaer condition *[55, Prop. 5.8] just if every section-retraction pair is an isomorphism.*

Combining these two conditions is equivalent to the following definition.

▶ **Definition 18.** *A category is* spacelike *if every idempotent is an identity.*

Lawvere has identified this condition as having particular importance in recognising petit toposes [77]. We will not use it much, as it restricts the dualities we wish to develop.

In the rest of this paper we will assume that our base categories $\mathcal{C}$ are Cauchy-complete, so that tiny objects coincide with representables.

4.3 Morphisms

The simplest kind of morphism between categories is a functor. Given a $f : \mathcal{C} \longrightarrow \mathcal{D}$ we can define a functor $f^* : [\mathcal{D}, \mathbf{Set}] \longrightarrow [\mathcal{C}, \mathbf{Set}]$ that takes $P : \mathcal{D} \longrightarrow \mathbf{Set}$ to $P \circ f : \mathcal{C} \longrightarrow \mathbf{Set}$. This functor has left and right adjoints, which are given by Kan extension [62, A4.1.4]:

$$[\mathcal{C}, \mathbf{Set}] \underset{f_!}{\overset{f_*}{\underset{\longleftarrow}{\overset{\longrightarrow}{\top \quad \top}}}} [\mathcal{D}, \mathbf{Set}] \qquad (f^*) \tag{10}$$

Therefore f^* preserves all limits and colimits, i.e. it is (co)continuous. In short, the presheaf construction gives a functor $[-, \mathbf{Set}] : \mathbf{Cat}_{\mathrm{cc}}^{\mathrm{op}} \longrightarrow \mathbf{PshCat}$, where $\mathbf{Cat}_{\mathrm{cc}}$ is the category of small Cauchy-complete categories and functors; and **PshCat** is the category of presheaf categories (over Cauchy-complete base categories) and (co)continuous functors.

Moreover, this functor is an equivalence. Given a presheaf category we can obtain its base as the subcategory of tiny objects [62, A1.1.10]. But how can we extract $f : \mathcal{C} \longrightarrow \mathcal{D}$ from any (co)continuous functor $f^* : [\mathcal{D}, \mathbf{Set}] \longrightarrow [\mathcal{C}, \mathbf{Set}]$? First, as presheaf categories are locally presentable, the adjoint functor theorem implies that f^* has left and right adjoints, as in (10) [1, §1.66]. This gives what topos theorists call an *essential geometric morphism*. Johnstone [62, §A4.1.5] shows that every such morphism is induced by a $f : \mathcal{C} \longrightarrow \mathcal{D}$, as $f_!$ preserves representables (when $\mathcal{D}$ is Cauchy-complete). We thus obtain a duality

$$\mathbf{Cat}_{\mathrm{cc}}^{\mathrm{op}} \simeq \mathbf{PshCat} \tag{11}$$

As with posets, functors here fail to preserve truth; for that we need a notion of openness.

▶ **Definition 19.** *$f : \mathcal{C} \longrightarrow \mathcal{D}$ is* open *just if $f^* : [\mathcal{D}, \mathbf{Set}] \longrightarrow [\mathcal{C}, \mathbf{Set}]$ preserves exponentials.*

▶ **Lemma 20.** *If $f : \mathcal{C} \longrightarrow \mathcal{D}$ is open then there is a natural isomorphism $\theta_w : [\![\varphi]\!]_w \cong [\![\varphi]\!]_{f(w)}$.*

Definition 19 is somewhat underwhelming, as it does not give explicit conditions that one can check – unlike Definition 3. However, obtaining such a description appears difficult.

Some information may be gleaned by considering $(f^*, f_*) : [\mathcal{C}, \mathbf{Set}] \longrightarrow [\mathcal{D}, \mathbf{Set}]$ as a *geometric morphism*. Such a morphism is *open* [60] [63, C3.1] just if both the canonical maps $f^*(c \Rightarrow d) \to f^*(c) \Rightarrow f^*(d)$ and $f^*(\Omega) \to \Omega$ are monic. Johnstone [63, C3.1] proves that (f^*, f_*) is open iff for any $\beta : f(w) \to v'$ in $\mathcal{D}$ there exists an $\alpha : w \to w'$ in $\mathcal{C}$ and a section-retraction pair $s : v' \to f(w')$ and $r : f(w') \to v'$ with $s \circ \beta = f(\alpha)$. This superficially seems like a categorification of Definition 3. However, it only guarantees that f^* is *sub-cartesian-closed*, whereas we need an isomorphism for Lemma 20 to hold.

A stronger condition is to ask that (f^*, f_*) be *locally connected*, i.e. that f^* commute with dependent products [63, C3.3]. All such morphisms are open geometric morphisms. This is stronger than what we need, but sufficient conditions on f can be given [63, C3.3.8].

Finally, an even stronger condition is to ask that (f^*, f_*) be *atomic*, i.e. that f^* is a *logical* functor. This means it preserves exponentials and the subobject classifier [63, A2.1, C3.5]. All atomic geometric morphisms are locally connected. This is again stronger than what we need, and a characterisation in terms of f is elusive: see MathOverflow [95].

It is easier to characterise when (f^*, f_*) is a *surjective geometric morphism*, i.e. when f^* is faithful [62, A2.4.6]. This happens exactly when f is *retractionally surjective*, i.e. whenever every $d \in \mathcal{D}$ is the retract of $f(c)$ for some $c \in \mathcal{C}$ [62, A2.4.7]. If $\mathcal{D}$ satisfies the Hemelaer condition this reduces to f being essentially surjective.

Write $\mathcal{C} \vDash \varphi$ to mean that $[\![\varphi]\!]_w$ is non-empty for any $w \in \mathcal{C}$ and any interpretation of $[\![p]\!]$.

▶ **Lemma 21.** *Let $f : \mathcal{C} \to \mathcal{D}$ be open and retractionally surjective. If $\mathcal{C} \vDash \varphi$ then $\mathcal{D} \vDash \varphi$.*

We may thus restrict the duality (11) to dualities

$$\mathbf{Cat}_{\mathrm{cc,\ open}}^{\mathrm{op}} \simeq \mathbf{PshCat}_{\Rightarrow} \qquad\qquad \mathbf{Cat}_{\mathrm{cc,\ open,\ rs}}^{\mathrm{op}} \simeq \mathbf{PshCat}_{\Rightarrow,\mathrm{f}} \tag{12}$$

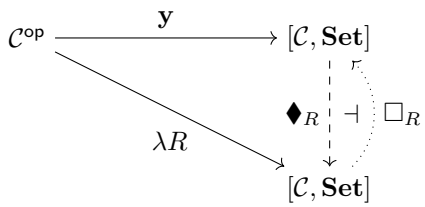
In the first instance the category to the left of $\simeq$ is that of small Cauchy-complete categories and open functors; and to the right of $\simeq$ it is presheaf categories and (co)complete, cartesian closed functors. In the second instance the category to the left of $\simeq$ is that of small Cauchy-complete categories and open, retractionally surjective functors; and to the right of $\simeq$ it is presheaf categories and (co)complete, faithful, cartesian closed functors.

5 Modal Logic II

To make a two-dimensional Kripke semantics for modal logic we have to categorify relations. We took the first step by considering bimodules, i.e. information-order-respecting relations. The second step can be taken by replacing 2 with **Set**; this leads us to the notion of a relation between categories, also known as a *profunctor* or *distributor* [9] [17, §7].

▶ **Definition 22.** *A* profunctor $R : \mathcal{C} \nrightarrow \mathcal{D}$ *is a functor* $R : \mathcal{C}^{\mathsf{op}} \times \mathcal{D} \to \mathbf{Set}$.

To formulate a two-dimensional Kripke semantics for modal logic we replace modal Kripke frames with a small Cauchy-complete category $\mathcal{C}$ with an (endo)profunctor $R : \mathcal{C}^{\mathsf{op}} \times \mathcal{C} \to \mathbf{Set}$. To obtain the modalities we can now play the same trick: putting $\lambda R : \mathcal{C}^{\mathsf{op}} \to [\mathcal{C}, \mathbf{Set}]$ into (9) we canonically obtain the following diagram by Kan extension:



(13)

Conversely, any adjunction $\blacklozenge \dashv \Box$ on $[\mathcal{C}, \mathbf{Set}]$ corresponds to the (endo)profunctor on $\mathcal{C}$ given by $(c_1, c_2) \mapsto \mathrm{Hom}_{\mathcal{C}}(c_1, \blacklozenge c_2)$.

We may then define $[\![\blacklozenge\varphi]\!] \stackrel{\text{def}}{=} \blacklozenge_R [\![\varphi]\!] : \mathcal{C} \longrightarrow \mathbf{Set}$ and $[\![\Box\varphi]\!] \stackrel{\text{def}}{=} \Box_R [\![\varphi]\!] : \mathcal{C} \longrightarrow \mathbf{Set}$. It is worth unfolding what a proof of $\Box\varphi$ at a world w is to obtain an explicit description:

$$[\![\Box\varphi]\!]_w = (\Box_R [\![\varphi]\!])(w) = \mathrm{Hom}_{[\mathcal{C},\mathbf{Set}]}(\lambda R(w), [\![\varphi]\!]) = \mathrm{Hom}_{[\mathcal{C},\mathbf{Set}]}(R(w,-), [\![\varphi]\!]) \tag{14}$$

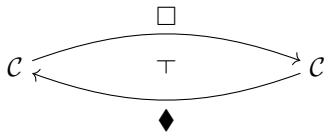
Thus, a proof that φ holds at w is a natural transformation $\alpha : R(w,-) \Rightarrow [\![\varphi]\!]$. This has the expected shape of Kripke semantics for $\Box$: for each $v \in \mathcal{C}$ and proof $x \in R(w,v)$ that v is accessible from w it gives us a proof $\alpha_v(x) \in [\![\varphi]\!]_v$ that φ holds at v.

It is a little harder to see what a proof of $\blacklozenge\varphi$ at a world w is. It becomes more perspicuous if we use the coend formula for the left Kan extension [74, §2.3]:

$$[\![\blacklozenge\varphi]\!] = \lambda R_! [\![\varphi]\!] \cong \int^{v \in \mathcal{C}} \mathrm{Hom}_{[\mathcal{C},\mathbf{Set}]}(\mathbf{y}(v), [\![\varphi]\!]) \times \lambda R(v) \cong \int^{v \in \mathcal{C}} [\![\varphi]\!]_v \times R(v,-) \tag{15}$$

Hence, a proof that $\blacklozenge\varphi$ holds at w consists of a world $v \in \mathcal{C}$, a proof that $R(v,w)$, and a proof that φ holds at v – which is exactly what one would expect. The difference is that the coend quotients some of these pairs according to the action of $\mathcal{C}$. See Mac Lane and Moerdijk [76, §VII.2] for a textbook exposition on why this is a tensor product of $[\![\varphi]\!]$ and λR.

How well does this fit the categorical semantics of modal logic? As with intuitionistic modal logic, there is also a number of proposals of what that might be. A fairly recent idea is to define it as the semantics of a *Fitch-style calculus*, as studied by Clouston [25]. This is exactly a bicartesian closed category $\mathcal{C}$ equipped with an adjunction:



(16)

The left adjoint $\blacklozenge$ is often written as lock. It does not commonly appear as a modality, but as an operator on contexts that corresponds to "opening a box" in Fitch-style natural deduction [59, §5.4]. The modality $\Box$ is a right adjoint, so that it automatically preserves all limits,

including products. This idea has proven remarkably robust: variations on it have worked well for modal dependent type theories [13, 51, 52, 50, 49]. The fact that an adjunction on a presheaf category corresponds precisely to a two-dimensional Kripke semantics is further evidence that this is the correct notion of categorical model of modal logic.

Finally, note that (14) and (15) look suspiciously similar to the modal structure of Normalization-by-Evaluation models for modal type theories. This is explicitly visible in the paper by Valliappan et al. [97, §2], and also implicitly present in the paper by Gratzer [48].

5.1 Morphisms

Define the category **Prof** to have as objects profunctors. A morphism $(f, g, \alpha) : R \to S$ from $R : \mathcal{C} \nrightarrow \mathcal{D}$ to $S : \mathcal{C}' \nrightarrow \mathcal{D}'$ consists of functors $f : \mathcal{C} \longrightarrow \mathcal{C}'$ and $g : \mathcal{D} \longrightarrow \mathcal{D}'$, and a natural transformation $\alpha : R(-,-) \Rightarrow S(f(-), g(-))$. The subcategory **EProf** consists of endoprofunctors $R : \mathcal{C} \nrightarrow \mathcal{C}$, and triples of the form (f, f, α). I will synecdochically refer to $\alpha : R(-,-) \Rightarrow S(f(-), f(-))$ as a morphism of **EProf**. Thus, objects are two-dimensional Kripke frames, and morphisms are functors that proof-relevantly preserve the relation.

▶ **Lemma 23.** *Morphisms of endoprofunctors $\alpha : R(-,-) \Rightarrow S(f(-), f(-))$ are in bijection with natural transformations $\gamma : f^* \Box_S \Rightarrow \Box_R f^*$.*

Proof. Unfolding the definitions, $\gamma : \mathrm{Hom}(S(f(-),-),-) \Rightarrow \mathrm{Hom}(R(-,-), f^*(-))$. As $f_! \dashv f^*$ this is exactly a transformation $\mathrm{Hom}(S(f(-),-),-) \Rightarrow \mathrm{Hom}\Big(f_! R(-,-),-\Big)$. By the Yoneda lemma, any such transformation arises by precomposition with a unique transformation $f_! R(-,-) \Rightarrow S(f(-),-)$. By $f_! \dashv f^*$ again, this uniquely corresponds to a transformation $\alpha : R(-,-) \Rightarrow f^* S(f(-),-) = S(f(-), f(-))$. ◀

We thus obtain a duality

$$\mathbf{EProf}_{\mathrm{cc}}^{\mathrm{op}} \simeq \mathbf{PshCatO} \tag{17}$$

where **PshCatO** is the category of presheaf categories $[\mathcal{C}, \mathbf{Set}]$ equipped with a continuous $\Box : [\mathcal{C}, \mathbf{Set}] \longrightarrow [\mathcal{C}, \mathbf{Set}]$. Note that, as presheaf categories are locally finitely presentable, $\Box$ always has a left adjoint $\blacklozenge$. Thus, the objects are categorical models of modal logic. Morphisms are pairs (f, γ) of a (co)continuous $f : \mathcal{C} \longrightarrow \mathcal{D}$ and a natural transformation $\gamma : f^* \Box \Rightarrow \Box f^*$.

As before, open functors do not preserve truth; for that we need a notion of modal openness. Let $\alpha : R(-,-) \Rightarrow S(f(-), f(-))$. As pointed out in the proof of Lemma 23 such an α uniquely corresponds to a transformation $t_\alpha : f_! R(-,-) \Rightarrow S(f(-),-)$. Its components

$$t_{\alpha,c,v} : \int^{w \in V} R(c,w) \times \mathrm{Hom}_{\mathcal{D}}(f(w), v) \to S(f(c), v)$$

map $x \in R(c,w)$ and $k : f(w) \to v$ to $S(\mathsf{id}_{f(c)}, k)(\alpha_{c,v}(x))$. We can then say that

▶ **Definition 24.** $\alpha : R(-,-) \Rightarrow S(f(-), f(-))$ *is* modally open *just if t_α is an isomorphism.*

This asks that for every proof $y \in S(f(c), v)$ we should be able to find an object $w \in \mathcal{C}$, a proof $x \in R(c,w)$, and a morphism $k : f(w) \to v$, so that $y = S(\mathsf{id}_{f(c)}, k)(\alpha_{c,v}(x))$. This is clearly a categorification of Definition 10, and leads to the following lemma:

▶ **Lemma 25.** *α is modally open iff the corresponding $f^* \Box_S \Rightarrow \Box_R f^*$ is an isomorphism.*

Proof. The proof of Lemma 23 precomposes with t_α to get γ. Thus γ is iso iff t_α is. ◀

Thus, the duality (17) may be restricted to dualities between the wide subcategories

$$\mathbf{EProf}^{\mathrm{op}}_{\mathrm{cc,\ moo}} \simeq \mathbf{PshCatO}_{\Rightarrow o} \qquad \mathbf{EProf}^{\mathrm{op}}_{\mathrm{cc,\ moo,\ rs}} \simeq \mathbf{PshCatO}_{\Rightarrow of} \tag{18}$$

The morphisms to the left of $\simeq$ are modally open, open maps (resp. and retractionally surjective); and the morphisms to the right of $\simeq$ are (f, γ) where f is cartesian closed (resp. and faithful) and $\gamma : f^*\Box \cong \Box f^*$ is a natural isomorphism.

6 Other related work

Perhaps the work most closely related to this paper is that on *Kripke-style lambda models* by Mitchell and Moggi [79]. These amount to elaborating the first-order definitions of applicative structure and λ-model in the internal language of a presheaf category, with the base category being a partial order. In practice this means that the interpretation of function types is only a subfunctor of the exponential of presheaves [79, §8]. However, Mitchell and Moggi prove that these models are sound and complete for the $(\times\ \to)$ fragment, even in the presence of empty types. They also use some general theorems about open geometric morphisms to prove that any cartesian closed category can be presented as such a model.

Another piece of work that bears kinship with the present one is Hermida's fibrational account of relational modalities [56]. Hermida shows that both the relational modalities $\Diamond$ and $\Box$ can be obtained canonically as extensions of predicate logic to relations, with the modalities arising as compositions of adjoints. The black diamond $\blacklozenge$ makes a brief cameo as the induced left adjoint to $\Box$, as does the dual black box [56, §3.3]. While the decompositions obtained by Hermida seem more refined than the results here, Kan extension does not make an explicit appearance. As such, the relationship to the present work is yet to be determined.

Awodey and Rabe [6] give a Kripke semantics for extensional Martin-Löf type theory (MLTT), in which contexts are posets and types are presheaves over them. They use topos-theoretic machinery to prove that every locally cartesian closed category can be embedded in a presheaf category over a poset; this result seems similar to one of Mitchell and Moggi, but the proof appears entirely different. As a consequence, they show that presheaf categories over posets form a complete class of models for extensional MLTT, in fact a subclass of locally cartesian closed categories.

Alechina et al. [2] present dualities between Kripke and algebraic semantics for constructive S4 and propositional lax logic. Their interpretation of $\Box$ follows that of Plotkin, Stirling and Simpson [85, 92].

Ghilardi and Meloni [42] explore a presheaf-like interpretation of (predicate) modal logic, which is similar to ours, albeit non-proof-relevant. They work over the identity profunctor $\mathrm{Hom}(-,-)$. They are hence forced to weaken the definition of presheaf. See also [43, 44].

Awodey, Kishida and Kotzsch [5] give a topos-theoretic semantics for a higher-order version of intuitionistic S4 modal logic. They also briefly survey much previous work on presheaf-based and topos-theoretic semantics for first-order modal logic. Their work is not proof-relevant.

Finally, there is clear methodological similarity between the results obtained here and the results of Winskel and collaborators on open maps and bisimulation [65, 22]. One central difference is that Winskel et al. are mainly concerned with open maps between presheaves themselves, whereas I only consider open maps between (two-dimensional) frames.

References

1 Jiří Adámek and Jiří Rosický. *Locally Presentable and Accessible Categories*. Cambridge University Press, 1994. doi:10.1017/CBO9780511600579.

2 Natasha Alechina, Michael Mendler, Valeria de Paiva, and Eike Ritter. Categorical and Kripke semantics for constructive S4 modal logic. *Computer Science Logic*, pages 292–307, 2001. doi:10.1007/3-540-44802-0_21.

3 Mathieu Anel and André Joyal. Topo-logie. In Mathieu Anel and Gabriel Catren, editors, *New Spaces in Mathematics*, pages 155–257. Cambridge University Press, 2021. doi:10.1017/9781108854429.007.

4 Steve Awodey. *Category Theory*. Oxford Logic Guides. Oxford University Press, 2010.

5 Steve Awodey, Kohei Kishida, and Hans-Christoph Kotzsch. Topos Semantics for Higher-Order Modal Logic. *Logique et Analyse*, 228:591–636, 2014. doi:10.2143/LEA.228.0.3078176.

6 Steve Awodey and Florian Rabe. Kripke Semantics for Martin-Löf's Extensional Type Theory. *Logical Methods in Computer Science*, 7(3):1–34, 2011. doi:10.2168/LMCS-7(3:18)2011.

7 John C. Baez and James Dolan. Categorification. In Ezra Getzler and Mikhail Kapranov, editors, *Higher Category Theory*, number 230 in Contemporary Mathematics. American Mathematical Society, 1998. eprint: math/9802029. doi:10.48550/arXiv.math/9802029.

8 B. Banaschewski and G. Bruns. The fundamental duality of partially ordered sets. *Order*, 5(1), 1988. doi:10.1007/BF00143898.

9 J. Bénabou. Distributors at work, 2000. Notes of Thomas Streicher based on lectures of Jean Bénabou. URL: https://www2.mathematik.tu-darmstadt.de/~streicher/FIBR/DiWo.pdf.

10 Evert Willem Beth. *The Foundations of Mathematics*. North-Holland Pub. Co., 1959.

11 Guram Bezhanishvili and Wesley H. Holliday. A semantic hierarchy for intuitionistic logic. *Indagationes Mathematicae*, 30(3):403–469, 2019. doi:10.1016/j.indag.2019.01.001.

12 Nick Bezhanishvili. *Lattices of intermediate and cylindric modal logics*. PhD thesis, University of Amsterdam, 2006. Report number DS-2006-02. URL: https://eprints.illc.uva.nl/id/eprint/2049/.

13 Lars Birkedal, Ranald Clouston, Bassel Mannaa, Rasmus Ejlers Møgelberg, Andrew M. Pitts, and Bas Spitters. Modal dependent type theory and dependent right adjoints. *Mathematical Structures in Computer Science*, 30(2):118–138, 2020. doi:10.1017/S0960129519000197.

14 Lars Birkedal, Rasmus 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.

15 Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2001. doi:10.1017/CBO9781107050884.

16 Francis Borceux. *Handbook of Categorical Algebra*, volume 3 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 1994.

17 Francis Borceux. *Handbook of Categorical Algebra*, volume 1 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 1994.

18 Francis Borceux and Dominique Dejean. Cauchy completion in category theory. *Cahiers de Topologie et Géométrie Différentielle Catégoriques*, 27(2):133–146, 1986. URL: http://www.numdam.org/item/?id=CTGDC_1986__27_2_133_0.

19 Milan Božić and Kosta Došen. Models for normal intuitionistic modal logics. *Studia Logica*, 43(3):217–245, 1984. doi:10.1007/BF02429840.

20 Marta Bunge. *Categories of Set-Valued Functors*. PhD thesis, University of Pennsylvania, 1966.

21 Gian Luca Cattani and Glynn Winskel. Presheaf models for concurrency. In Dirk Dalen and Marc Bezem, editors, *Computer Science Logic*, volume 1258 of *Lecture Notes in Computer Science*, pages 58–75, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg. doi:10.1007/3-540-63172-0_32.

22 Gian Luca Cattani and Glynn Winskel. Profunctors, open maps and bisimulation. *Mathematical Structures in Computer Science*, 15(3):553–614, 2005. doi:10.1017/S0960129505004718.

23 Alexander Chagrov and Michael Zakharyaschev. *Modal Logic*. Number 35 in Oxford Logic Guides. Oxford University Press, 1996. doi:10.1093/oso/9780198537793.001.0001.

24 Denis-Charles Cisinski. *Higher Categories and Homotopical Algebra*. Cambridge University Press, 2019. doi:10.1017/9781108588737.

25 Ranald Clouston. Fitch-Style Modal Lambda Calculi. In Christel Baier and Ugo Dal Lago, editors, *Foundations of Software Science and Computation Structures*, volume 10803 of *Lecture Notes in Computer Science*, pages 258–275, Cham, 2018. Springer International Publishing. doi:10.1007/978-3-319-89366-2_14.

26 Roy L. Crole. *Categories for Types*. Cambridge University Press, 1993. doi:10.1017/CBO9781139172707.

27 Anupam Das and Sonia Marin. On intuitionistic diamonds (and lack thereof). In Revantha Ramanayake and Josef Urban, editors, *Automated Reasoning with Analytic Tableaux and Related Methods*, volume 14278 of *Lecture Notes in Computer Science*, pages 283–301. Springer Nature Switzerland, 2023. doi:10.1007/978-3-031-43513-3_16.

28 B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 2nd edition, 2002. doi:10.1017/CBO9780511809088.

29 D.H.J. De Jongh and A.S. Troelstra. On the connection of partially ordered sets with some pseudo-boolean algebras. *Indagationes Mathematicae (Proceedings)*, 69:317–329, 1966. doi:10.1016/S1385-7258(66)50036-1.

30 Wojciech Dzik, Jouni Järvinen, and Michiro Kondo. Intuitionistic propositional logic with Galois connections. *Logic Journal of IGPL*, 18(6):837–858, 2010. doi:10.1093/jigpal/jzp057.

31 M. Erné. The ABC of order and topology. In H. Herrlich and H. E. Porst, editors, *Category Theory at Work*, Research and Exposition in Mathematics, pages 57–83. Heldermann Verlag, 1991. URL: https://www.heldermann.de/R&E/RAE18/ctw05.pdf.

32 Leo Esakia. *Heyting Algebras: Duality Theory*, volume 50 of *Trends in Logic*. Springer International Publishing, 2019. doi:10.1007/978-3-030-12096-2.

33 W. B. Ewald. Intuitionistic Tense and Modal Logic. *The Journal of Symbolic Logic*, 51(1):166–179, 1986. doi:10.2307/2273953.

34 M. Fiore, G. Plotkin, and D. Turi. Abstract syntax and variable binding. In *Proceedings. 14th Symposium on Logic in Computer Science (Cat. No. PR00158)*, pages 193–202, Trento, Italy, 1999. IEEE Computer Society Press. doi:10.1109/LICS.1999.782615.

35 Marcelo Fiore. Second-Order and Dependently-Sorted Abstract Syntax. In *2008 23rd Annual IEEE Symposium on Logic in Computer Science*, pages 57–68. IEEE, 2008. doi:10.1109/LICS.2008.38.

36 Marcelo Fiore and Chung-Kil Hur. Second-Order Equational Logic (Extended Abstract). In Anuj Dawar and Helmut Veith, editors, *Computer Science Logic*, volume 6247 of *Lecture Notes in Computer Science*, pages 320–335. Springer Berlin Heidelberg, 2010. doi:10.1007/978-3-642-15205-4_26.

37 Marcelo Fiore and Ola Mahmoud. Second-Order Algebraic Theories: (Extended Abstract). In Petr Hliněný and Antonín Kučera, editors, *Mathematical Foundations of Computer Science 2010*, volume 6281 of *Lecture Notes in Computer Science*, pages 368–380. Springer Berlin Heidelberg, 2010. doi:10.1007/978-3-642-15155-2_33.

38 Gisèle Fischer Servi. Semantics for a class of intuitionistic modal calculi. In *Italian Studies in the Philosophy of Science*, volume 47 of *Boston Studies in the Philosophy of Science*, pages 59–72. Springer Netherlands, 1980. doi:10.1007/978-94-009-8937-5_5.

39 Melvin Fitting. *Intuitionistic Logic, Model Theory and Forcing*. Studies in Logic and the Foundation of Mathematics. North-Holland, 1969.

40 Mai Gehrke. Stone duality, topological algebra, and recognition. *Journal of Pure and Applied Algebra*, 220(7):2711–2747, 2016. doi:10.1016/j.jpaa.2015.12.007.

41 Mai Gehrke and Sam van Gool. *Topological Duality for Distributive Lattices: Theory and Applications*. Number 61 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2024. arXiv:2203.03286.

42 S. Ghilardi and G. C. Meloni. Modal and tense predicate logic: Models in presheaves and categorical conceptualization. In Francis Borceux, editor, *Categorical Algebra and its Applications*, volume 1348 of *Lecture Notes in Mathematics*, pages 130–142. Springer Berlin Heidelberg, 1988. doi:10.1007/BFb0081355.

43 Silvio Ghilardi. Presheaf semantics and independence results for some non-classical first-order logics. *Archive for Mathematical Logic*, 29(2):125–136, 1989. doi:10.1007/BF01620621.

44 Silvio Ghilardi. Incompleteness Results in Kripke Semantics. *The Journal of Symbolic Logic*, 56(2):517–538, 1991. doi:10.2307/2274697.

45 Robert Goldblatt. *Metamathematics of Modal Logic*. PhD Thesis, Victoria University, Wellington, 1974.

46 Robert Goldblatt. Mathematical modal logic: A view of its evolution. In Dov M. Gabbay, editor, *Handbook of the History of Logic*, volume 7. Elsevier, 2006. doi:10.1016/S1570-8683(03)00008-9.

47 Valentin Goranko and Antje Rumberg. Temporal Logic. In Edward N. Zalta and Uri Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Fall 2023 edition, 2023.

48 Daniel Gratzer. Normalization for Multimodal Type Theory. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 1–13. Association for Computing Machinery, 2022. doi:10.1145/3531130.3532398.

49 Daniel Gratzer. *Syntax and semantics of modal type theory*. PhD thesis, Aarhus University, 2023.

50 Daniel Gratzer, Evan Cavallo, G. A. Kavvos, Adrien Guatto, and Lars Birkedal. Modalities and Parametric Adjoints. *ACM Transactions on Computational Logic*, 23(3):1–29, 2022. doi:10.1145/3514241.

51 Daniel Gratzer, G. A. Kavvos, Andreas Nuyts, and Lars Birkedal. Multimodal Dependent Type Theory. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 492–506. Association for Computing Machinery, 2020. doi:10.1145/3373718.3394736.

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

53 Makoto Hamana. Free Σ-Monoids: A Higher-Order Syntax with Metavariables. In Wei-Ngan Chin, editor, *Programming Languages and Systems*, volume 3302 of *Lecture Notes in Computer Science*, pages 348–363. Springer Berlin Heidelberg, 2004. doi:10.1007/978-3-540-30477-7_23.

54 Yasusi Hasimoto. Heyting Algebras with Operators. *Mathematical Logic Quarterly*, 47(2):187–196, 2001. doi:10.1002/1521-3870(200105)47:2<187::AID-MALQ187>3.0.CO;2-J.

55 Jens Hemelaer. Toposes over which essential implies locally connected. *Cahiers de Topologie et Géométrie Différentielle Catégoriques*, LXIII(4):425–451, 2022. arXiv:2204.02749.

56 Claudio Hermida. A categorical outlook on relational modalities and simulations. *Information and Computation*, 209(12):1505–1517, 2011. doi:10.1016/j.ic.2010.09.009.

57 Martin Hofmann. Syntax and Semantics of Dependent Types. In Andrew M. Pitts and P. Dybjer, editors, *Semantics and Logics of Computation*, pages 79–130. Cambridge University Press, 1997. doi:10.1017/CBO9780511526619.004.

58 Martin Hofmann. Semantical analysis of higher-order abstract syntax. In *Proceedings. 14th Symposium on Logic in Computer Science (Cat. No. PR00158)*, pages 204–213. IEEE Computer Society Press, 1999. doi:10.1109/LICS.1999.782616.

59 Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, 2nd edition, 2004.

60 Peter T. Johnstone. Open maps of toposes. *Manuscripta Mathematica*, 31(1):217–247, 1980. doi:10.1007/BF01303275.

61 Peter T. Johnstone. *Stone Spaces*. Number 3 in Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1982.

62 Peter T. Johnstone. *Sketches of an Elephant: A Topos Theory Compendium*, volume 1. Oxford University Press, 2002.

63 Peter T. Johnstone. *Sketches of an Elephant: A Topos Theory Compendium*, volume 2. Oxford University Press, 2002.

64 Bjarni Jonnson and Alfred Tarski. Boolean algebras with operators. part II. *American Journal of Mathematics*, 74(1):127, 1952. doi:10.2307/2372074.

65 André Joyal, Mogens Nielsen, and Glynn Winskel. Bisimulation from Open Maps. *Information and Computation*, 127(2):164–185, 1996. doi:10.1006/inco.1996.0057.

66 G. Max Kelly. Basic concepts of enriched category theory. *Reprints in Theory and Applications of Categories*, 10:1–136, 2005. Originally published as: Cambridge University Press, Lecture Notes in Mathematics 64, 1982. URL: http://www.tac.mta.ca/tac/reprints/articles/10/tr10abs.html.

67 Kohei Kishida. Categories and modalities. In Elaine Landry, editor, *Categories for the Working Philosopher*. Oxford University Press, 2018. doi:10.1093/oso/9780198748991.003.0009.

68 Saul Kripke. Semantical Considerations on Modal Logic. *Acta Philosophica Fennica*, 16:83–94, 1963. URL: https://philpapers.org/rec/KRISCO.

69 Saul A. Kripke. Semantical Analysis of Modal Logic I. Normal Modal Propositional Calculi. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 9(5-6):67–96, 1963. doi:10.1002/malq.19630090502.

70 Saul A. Kripke. Semantical Analysis of Modal Logic II. Non-Normal Modal Propositional Calculi. In Alfred Tarski, Leon Henkin, and J. W. Addison, editors, *The theory of models, Proceedings of the 1963 International Symposium at Berkeley*, pages 206–220. North-Holland, 1965.

71 Joachim Lambek. From lambda calculus to cartesian closed categories. In Jonathan P. Seldin and J. Roger Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, pages 376–402. Academic Press, 1980.

72 Joachim Lambek and Philip J. Scott. *Introduction to Higher-Order Categorical Logic*. Number 7 in Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1988.

73 F. William Lawvere. Metric spaces, generalized logic, and closed categories. *Rendiconti del Seminario Matematico e Fisico di Milano*, 43(1):135–166, 1973. doi:10.1007/BF02924844.

74 Fosco Loregian. *(Co)end Calculus*. Cambridge University Press, 2021. doi:10.1017/9781108778657.

75 Saunders Mac Lane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer New York, 2 edition, 1978. doi:10.1007/978-1-4757-4721-8.

76 Saunders Mac Lane and Ieke Moerdijk. *Sheaves in Geometry and Logic: A First Introduction to Topos Theory*. Universitext. Springer New York, 1994. doi:10.1007/978-1-4612-0927-0.

77 Colin McLarty. Every Grothendieck topos has a one-way site. *Theory and Applications of Categories*, 16(5):123–126, 2006. URL: http://www.tac.mta.ca/tac/volumes/16/5/16-05abs.html.

78 M. Menni and C. Smith. Modes of adjointness. *Journal of Philosophical Logic*, 43(2):365–391, 2014. doi:10.1007/s10992-012-9266-y.

79 John C. Mitchell and Eugenio Moggi. Kripke-style models for typed lambda calculus. *Annals of Pure and Applied Logic*, 51(1-2):99–124, 1991. doi:10.1016/0168-0072(91)90067-V.

80 Mogens Nielsen, Gordon Plotkin, and Glynn Winskel. Petri nets, event structures and domains, Part I. *Theoretical Computer Science*, 13(1):85–108, 1981. doi:10.1016/0304-3975(81)90112-2.

81 F. J. Oles. Type Algebras, Functor Categories, and Block Structure. In Maurice Nivat and John C. Reynolds, editors, *Algebraic Methods in Semantics*, pages 543–573. Cambridge University Press, 1985.

82 Frank J. Oles. Functor Categories and Store Shapes. In Peter W. O'Hearn and Robert D. Tennent, editors, *Algol-like Languages*, volume 2, pages 3–12. Birkhäuser Boston, Boston, MA, 1997. doi:10.1007/978-1-4757-3851-3_1.

83 Ewa Orłowska and Ingrid Rewitzky. Discrete Dualities for Heyting algebras with Operators. *Fundamenta Informaticae*, 81(1):275–295, 2007.

84 Jorge Picado and Aleš Pultr. *Frames and Locales: Topology without points*. Frontiers in Mathematics. Springer Basel, 2012. doi:10.1007/978-3-0348-0154-6.

85 Gordon D. Plotkin and Colin Stirling. A framework for intuitionistic modal logics. In Joseph Y. Halpern, editor, *Proceedings of the 1st Conference on Theoretical Aspects of Reasoning about Knowledge, Monterey, CA, USA, March 1986*. Morgan Kaufmann, 1986.

86 George N. Raney. Completely distributive complete lattices. *Proceedings of the American Mathematical Society*, 3(5):677–680, 1952. doi:10.1090/S0002-9939-1952-0052392-3.

87 Emily Riehl. *Categorical homotopy theory*, volume 24 of *New Mathematical Monographs*. Cambridge University Press, 2014. doi:10.1017/CBO9781107261457.

88 Emily Riehl. *Category Theory in Context*. Dover Publications, 2016. URL: http://www.math.jhu.edu/~eriehl/context.pdf.

89 Mehrnoosh Sadrzadeh and Roy Dyckhoff. Positive logic with adjoint modalities: proof theory, semantics, and reasoning about information. *The Review of Symbolic Logic*, 3(3):351–373, 2010. doi:10.1017/S1755020310000134.

90 Davide Sangiorgi. On the origins of bisimulation and coinduction. *ACM Transactions on Programming Languages and Systems*, 31(4):1–41, 2009. doi:10.1145/1516507.1516510.

91 Krister Segerberg. Decidability of S4.1. *Theoria*, 34(1):7–20, 1968. doi:10.1111/j.1755-2567.1968.tb00335.x.

92 Alex K. Simpson. *The Proof Theory and Semantics of Intuitionistic Modal Logic*. PhD thesis, The University of Edinburgh, 1994. URL: http://hdl.handle.net/1842/407.

93 Morten Heine Sørensen and Pawel Urzyczyn. *Lectures on the Curry-Howard Isomorphism*. Elsevier, 2006.

94 V. H. Sotirov. Modal theories with intuitionistic logic. In *Mathematical Logic: Proceedings of the Conference on Mathematical Logic, Dedicated to the Memory of A.A. Markov (1903-1979): Sofia, September 22-23, 1980*. Bulgarian Academy of Sciences, 1984.

95 David Spivak. Conditions for a functor to induce a logical functor between presheaf toposes? MathOverflow, 2016. URL: https://mathoverflow.net/q/253878.

96 S. K. Thomason. Categories of frames for modal logic. *The Journal of Symbolic Logic*, 40(3):439–442, 1975. doi:10.2307/2272167.

97 Nachiappan Valliappan, Fabian Ruch, and Carlos Tomé Cortiñas. Normalization for Fitch-style Modal Calculi. *Proceedings of the ACM on Programming Languages*, 6(ICFP):772–798, 2022. doi:10.1145/3547649.

98 Johan van Benthem. *Modal logic for open minds*. Number 199 in CSLI lecture notes. Center for the Study of Language and Information, Stanford, California, 2010.

99 Philip Wadler. Propositions as types. *Communications of the ACM*, 58(12):75–84, 2015. doi:10.1145/2699407.

100 Glynn Winskel. Prime algebraicity. *Theoretical Computer Science*, 410(41):4160–4168, 2009. doi:10.1016/j.tcs.2009.06.015.

101 F. Wolter and M. Zakharyaschev. The relation between intuitionistic and classical modal logics. *Algebra and Logic*, 36(2):73–92, 1997. doi:10.1007/BF02672476.

102 Frank Wolter and Michael Zakharyaschev. Intuitionistic Modal Logic. In Andrea Cantini, Ettore Casari, and Pierluigi Minari, editors, *Logic and Foundations of Mathematics: Selected Contributed Papers of the Tenth International Congress of Logic, Methodology and Philosophy of Science, Florence, August 1995*, pages 227–238. Springer Netherlands, Dordrecht, 1999. doi:10.1007/978-94-017-2109-7_17.

103 Frank Wolter and Michael Zakharyaschev. Intuitionistic Modal Logics as Fragments of Classical Bimodal Logics. In Ewa Orłowska, editor, *Logic at Work: Essays Dedicated to the Memory of Helena Rasiowa*, Studies in Fuzziness and Soft Computing, pages 168–186. Physica Heidelberg, 1999.

104 David Yetter. On right adjoints to exponential functors. *Journal of Pure and Applied Algebra*, 45(3):287–304, 1987. doi:10.1016/0022-4049(87)90077-6.

Adjoint Natural Deduction

Junyoung Jang
McGill University, Montreal, Canada

Sophia Roshal
Carnegie Mellon University, Pittsburgh, USA

Frank Pfenning
Carnegie Mellon University, Pittsburgh, USA

Brigitte Pientka
McGill University, Montreal, Canada

Abstract

Adjoint logic is a general approach to combining multiple logics with different structural properties, including linear, affine, strict, and (ordinary) intuitionistic logics, where each proposition has an intrinsic mode of truth. It has been defined in the form of a sequent calculus because the central concept of independence is most clearly understood in this form, and because it permits a proof of cut elimination following standard techniques.

In this paper we present a natural deduction formulation of adjoint logic and show how it is related to the sequent calculus. As a consequence, every provable proposition has a verification (sometimes called a long normal form). We also give a computational interpretation of adjoint logic in the form of a functional language and prove properties of computations that derive from the structure of modes, including freedom from garbage (for modes without weakening and contraction), strictness (for modes disallowing weakening), and erasure (based on a preorder between modes). Finally, we present a surprisingly subtle algorithm for type checking.

2012 ACM Subject Classification Theory of computation → Proof theory; Theory of computation → Linear logic

Keywords and phrases Substructural Logic, Type Systems, Functional Programming

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.15

Related Version *Full Version*: https://arxiv.org/abs/2402.01428

Funding *Junyoung Jang*: Fonds de recherche du Québec – Nature et Technologies (FRQNT)
Brigitte Pientka: Natural Sciences and Engineering Research Council of Canada (NSERC) and Fonds de recherche du Québec – Nature et Technologies (FRQNT)

1 Introduction

A *substructural logic* provides fine control over the use of assumptions during reasoning. It usually does so by denying the general sequent calculus rules of *contraction* (which permits an antecedent to be used more than once) and *weakening* (which permits an antecedent not to be used). Instead, these rules become available only for antecedents of the form $!A$. Ever since the inception of linear logic [23], researchers have found applications in programming languages, for example, to avoid garbage collection [24], soundness of imperative update [53], the chemical abstract machine [2], and session-typed communication [12, 54], to name just a few.

Besides linear logic, there are other substructural logics and type systems of interest. For example, *affine logic* denies general contraction but allows weakening and is the basis for the type system of Alms [51] (an affine functional language) and Rust [50] (an imperative language aimed at systems programming).

If we deny weakening but accept contraction we obtain *strict logic* (a variant of *relevance logic*) where every assumption must be used at least once. On the programming language side, this corresponds to *strictness*, which allows optimizations in otherwise nonstrict functional languages such as Haskell [39]. Interestingly, Church's original λI calculus [18] was also strict in this sense.

The question arises how we can combine such features, both in logics and in type systems. Recently, this question has been tackled through graded or quantitative type systems (see for example, [37, 5, 38, 17, 55, 1]). The essential idea is to track and reason explicitly about the usage of a given assumption through grades. This provides very fine-grained control and allows us to, for example, model linear, strict, and unrestricted usage of assumptions through graded modalities. In this paper, we pursue an alternative, taking a proof-theoretic view with the goal of building a computational interpretation. There are three possible options that emerge from existing proof-theoretic explorations that could serve as a foundation of such a computational interpretation. The first one is by *embedding*. For example, we can embed (structural) intuitionistic logic in linear logic writing $!A \multimap B$ for $A \to B$. Similarly, we can embed affine logic in linear logic by mapping hypotheses A to $A \,\&\, \mathbf{1}$ so they do not need to be used. The difficulties with such embeddings is that, often, they neither respect proof search properties such as focusing [4] nor do they achieve a desired computational interpretation.

A second approach is taken by *subexponential linear logic* [20, 41, 30] that defines multiple subexponential modalities $!^m A$, where each *mode* m has a specific set of structural properties. As in linear logic, all inferences are carried out on linear formulas, so while it resolves some of the issues with embeddings, it still requires frequent movement into the linear layer using explicit subexponentials.

We pursue a third approach, pioneered by Benton [7] who symmetrically combined (structural) intuitionistic logic with (purely) linear intuitionistic logic. He employs two adjoint modalities that switch between the two layers and works out the proof theoretic and categorical semantics. This approach has the advantage that one can natively reason and compute within the individual logics, so we preserve not only provability but the fine structure of proofs and proof reduction from each component. This has been generalized in prior work [49, 46] by incorporating from subexponential linear logic the idea to have a preorder between modes $m \geq k$ that must be compatible with the structural properties of m and k (explained in more detail in Section 2). This means we can now also model intuitionistic S4 [42] and lax logic [9], representing comonadic and monadic programming, respectively. We hence arrive at a unifying calculus firmly rooted in proof theory that is more general than previous graded modal type systems in that we can *construct* monads as well as comonads. We will briefly address dependently typed variations of the adjoint approach in Section 7.

Most substructural logics and many substructural type systems are most clearly formulated as sequent calculi. However, natural deduction has not only an important foundational role [22, 44, 21], it also has provided a simple and elegant notation for functional programs through the Curry-Howard correspondence [27]. We therefore develop a system of natural deduction for adjoint logic that, in a strong sense, corresponds to the original sequent formulation. It turns out to be surprisingly subtle because we have to manage not only the substructural properties that may be permitted or not, but also respect the preorder between modes. We show that our calculus satisfies some expected properties like *substitution* and has a natural notion of *verification* that corresponds to proofs in long normal form, satisfying a subformula property.

In order to illustrate computational properties, we give an abstract machine and show the consequences of the mode structure: freedom from garbage for linear modes (that is, modes admitting neither weakening nor contraction), strictness for modes that do not admit weakening, and erasure for modes that a final value may not depend on, based on the preorder of modes. We close with an algorithmic type checker for our language which, again, is surprisingly subtle.

2 Adjoint Sequent Calculus

We briefly review the adjoint sequent calculus from [46]. We start with a standard set of possibly substructural propositions, indexing each with a *mode of truth*, denoted by m, k, n, r. Propositions are perhaps best understood by using their linear meaning as a guide, so we uniformly use the notation of linear logic. Also, for programming convenience, we generalize the usual binary and nullary disjunction ($A \oplus B$ and $\mathbf{0}$) and conjunction ($A \,\&\, B$ and $\top$) by using labeled disjunction $\oplus\{\ell : A^\ell_m\}_{\ell\in L}$ and conjunction $\&\{\ell : A^\ell_m\}_{\ell\in L}$. From the linear logical perspective, these are *internal* and *external* choice, respectively; from the programming perspective they are *sums* and *products*. We write P_m for atomic propositions of mode m.

$$
\begin{array}{llll}
\text{Propositions} & A_m, B_m & ::= & P_m \mid A_m \multimap B_m \mid \&\{\ell : A^\ell_m\}_{\ell\in L} \mid \uparrow^m_k A_k \quad \text{(negative)} \\
 & & & \mid A_m \otimes B_m \mid \mathbf{1}_m \mid \oplus\{\ell : A^\ell_m\}_{\ell\in L} \mid \downarrow^n_m A_n \quad \text{(positive)} \\
\text{Contexts} & \Gamma & ::= & \cdot \mid \Gamma, x : A_m \quad \text{(unordered)}
\end{array}
$$

Each mode m comes with a set $\sigma(m) \subseteq \{\mathsf{W}, \mathsf{C}\}$ of *structural properties*, where W stands for weakening and C stands for contraction. We further have a preorder $m \geq r$ that specifies that a proof of the succedent C_r may depend on an antecedent A_m. This is enforced using the presupposition that in a sequent $\Gamma \vdash C_r$, every antecedent A_m in Γ must satisfy $m \geq r$, written as $\Gamma \geq r$. We have the additional stipulation of *monotonicity*, namely that $m \geq k$ implies $\sigma(m) \supseteq \sigma(k)$. This is required for cut elimination to hold. Furthermore, we presuppose that in $\uparrow^m_k A_k$ we have $m \geq k$ and for $\downarrow^n_m A_n$ we have $n \geq m$. Also, contexts may not have any repeated variables and we will implicitly apply variable renaming to maintain this presupposition. Finally, we abbreviate $\cdot, x : A$ as just $x : A$.

In preparation for natural deduction, instead of explicit rules of weakening and contraction (see [46] for such a system) we have a context merge operation $\Gamma_1 \,;\, \Gamma_2$. Since, as usual in the sequent calculus, we read the rules bottom-up, it actually describes a nondeterministic split of the context that is pervasive in the presentations of linear logic [4].

$$
\begin{array}{rcrcll}
(\Gamma_1, x : A_m) & ; & (\Gamma_2, x : A_m) & = & (\Gamma_1 \,;\, \Gamma_2), x : A_m & \text{provided } \mathsf{C} \in \sigma(m) \\
(\Gamma_1, x : A_m) & ; & \Gamma_2 & = & (\Gamma_1 \,;\, \Gamma_2), x : A_m & \text{provided } x \notin \mathrm{dom}(\Gamma_2) \\
\Gamma_1 & ; & (\Gamma_2, x : A_m) & = & (\Gamma_1 \,;\, \Gamma_2), x : A_m & \text{provided } x \notin \mathrm{dom}(\Gamma_1) \\
(\cdot) & ; & \Gamma_2 & = & \Gamma_2 & \\
\Gamma_1 & ; & (\cdot) & = & \Gamma_1 &
\end{array}
$$

Note that the context merge is a partial operation, which prevents, for example, the use of an antecedent without contraction in both premises of the $\otimes R$ rule.

The complete set of rules can be found in Figure 1. In the rules, we write Γ_{W} for a context in which weakening can be applied to every antecedent, that is, $\mathsf{W} \in \sigma(m)$ for every antecedent $x : A_m$. Also, as is often the case in presentations of the sequent calculus, we omit explicit variable names that tag antecedents. We only discuss the rules for $\downarrow^n_m A_n$ because they illustrate the combined reasoning about structural properties and modes.

$$\frac{}{\Gamma_{\mathsf{W}} \,;\, A_m \vdash A_m}\ \mathsf{id} \qquad \frac{\Gamma \geq m \geq r \quad \Gamma \vdash A_m \quad \Gamma', A_m \vdash C_r}{\Gamma \,;\, \Gamma' \vdash C_r}\ \mathsf{cut}$$

$$\frac{\Gamma, A_m \vdash B_m}{\Gamma \vdash A_m \multimap B_m} \multimap R \qquad \frac{\Gamma \geq m \quad \Gamma \vdash A_m \quad \Gamma', B_m \vdash C_r}{\Gamma \,;\, \Gamma' \,;\, A_m \multimap B_m \vdash C_r} \multimap L$$

$$\frac{\Gamma \vdash A^\ell_m \quad (\forall \ell \in L)}{\Gamma \vdash \&\{\ell : A^\ell_m\}_{\ell \in L}} \,\&R \qquad \frac{\Gamma, A^\ell_m \vdash C_r \quad (\ell \in L)}{\Gamma \,;\, \&\{\ell : A^\ell_m\}_{\ell \in L} \vdash C_r} \,\&L$$

$$\frac{\Gamma \vdash A_k}{\Gamma \vdash \uparrow^m_k A_k} \uparrow R \qquad \frac{k \geq r \quad \Gamma, A_k \vdash C_r}{\Gamma \,;\, \uparrow^m_k A_k \vdash C_r} \uparrow L$$

$$\frac{\Gamma \vdash A_m \quad \Gamma' \vdash B_m}{\Gamma \,;\, \Gamma' \vdash A_m \otimes B_m} \otimes R \qquad \frac{\Gamma, A_m, B_m \vdash C_r}{\Gamma \,;\, A_m \otimes B_m \vdash C_r} \otimes L$$

$$\frac{}{\Gamma_{\mathsf{W}} \vdash \mathbf{1}_m} \mathbf{1}R \qquad \frac{\Gamma \vdash C_r}{\Gamma \,;\, \mathbf{1}_m \vdash C_r} \mathbf{1}L$$

$$\frac{\Gamma \vdash A^\ell_m \quad (\ell \in L)}{\Gamma \vdash \oplus\{\ell : A^\ell_m\}_{\ell \in L}} \oplus R \qquad \frac{\Gamma, A^\ell_m \vdash C_r \quad (\forall \ell \in L)}{\Gamma \,;\, \oplus\{\ell : A^\ell_m\}_{\ell \in L} \vdash C_r} \oplus L$$

$$\frac{\Gamma' \geq n \quad \Gamma' \vdash A_n}{\Gamma_{\mathsf{W}} \,;\, \Gamma' \vdash \downarrow^n_m A_n} \downarrow R \qquad \frac{\Gamma, A_n \vdash C_r}{\Gamma \,;\, \downarrow^n_m A_n \vdash C_r} \downarrow L$$

Figure 1 Implicit Adjoint Sequent Calculus.

First, the $\downarrow R$ rule.

$$\frac{\Gamma' \geq n \quad \Gamma' \vdash A_n}{\Gamma_{\mathsf{W}} \,;\, \Gamma' \vdash \downarrow^n_m A_n} \downarrow R$$

Because we presuppose the conclusion is well-formed, we know $\Gamma_{\mathsf{W}} \,;\, \Gamma' \geq m$ since $\downarrow^n_m A_n$ has mode m. Again, by presupposition $n \geq m$ and we have to explicitly check that $\Gamma' \geq n$ because it doesn't follow from knowing that $\Gamma_{\mathsf{W}} \,;\, \Gamma' \geq m$. There may be some antecedents A_k in the conclusion such that $k \not\geq n$. If the mode k admits weakening, we can sort them into Γ_{W}. If it does not, then the rule is simply not applicable.

On to the $\downarrow L$ rule:

$$\frac{\Gamma, A_n \vdash C_r}{\Gamma \,;\, \downarrow^n_m A_n \vdash C_r} \downarrow L$$

By presupposition on the conclusion, we know $\Gamma \,;\, \downarrow^n_m A_n \geq r$ which means that $\Gamma \geq r$ and $m \geq r$. Since $n \geq m$ we have $n \geq r$ by transitivity, so $\Gamma, A_n \geq r$ and we do not need any explicit check. The formulation of the antecedents in the conclusion $\Gamma \,;\, \downarrow^n_m A_n$ means that if mode m admits contraction, then the antecedent $\downarrow^n_m A_n$ may also occur in Γ, that is, it may be preserved by the rule. If m does not admit contraction, this occurrence of $\downarrow^n_m A_n$ is not carried over to the premise.

This implicit sequent calculus satisfies the expected theorems, due to [49, 46] and, most closely reflecting the precise form of our formulation, [45]. They follow standard patterns, modulated by the substructural properties and the preorder on modes.

▶ **Theorem 1** (Admissibility of Weakening and Contraction). *The following are admissible:*

$$\frac{\Gamma_{\mathsf{W}} \geq m \quad \Gamma \vdash A_m}{\Gamma_{\mathsf{W}}\,;\Gamma \vdash A_m}\ \text{weaken} \qquad \frac{\mathsf{C} \in \sigma(m) \quad \Gamma, A_m, A_m \vdash C_r}{\Gamma, A_m \vdash C_r}\ \text{contract}$$

▶ **Theorem 2** (Admissibility of Cut and Identity).

(i) *In the system without cut, cut is admissible.*

(ii) *In the system with identity restricted to atoms P_m, the general identity is admissible.*

We call a proof *cut-free* if it does not contain cut and *long* if the identity is restricted to atomic propositions P. It is an immediate consequence of Theorem 2 that every derivable sequent has a long cut-free proof. The subformula property of cut-free proofs directly implies that a cut-free proof of a sequent $\Gamma_m \vdash A_m$ where all subformulas are of mode m is directly a proof in the logic captured by the mode m. Moreover, an arbitrary proof can be transformed into one of this form by cut elimination. These strong *conservative extension* properties are a hallmark of adjoint logic.

Since our main interest lies in natural deduction, we consider only three examples.

▶ **Example 3** (G3). We obtain the standard sequent calculus G3 [31] for intutionistic logic with a single mode U. All side conditions are automatically satisfied since $\mathsf{U} \geq \mathsf{U}$.

▶ **Example 4** (LNL and DILL). By specializing the rules to two modes, U and L with the order $\mathsf{U} > \mathsf{L}$, we obtain a minor variant of Linear/Non Linear Logic (LNL) in its *parsimonious presentation* [8]. Our notation is $FX = \downarrow^{\mathsf{U}}_{\mathsf{L}} X$ and $GA = \uparrow^{\mathsf{U}}_{\mathsf{L}} A$. Significant here is that we do not just model provability, but the exact structure of proofs except that our structural rules remain implicit.

We obtain the sequent calculus formulation of dual intuitionistic linear logic (DILL) [6, 15] by restricting the formulas of mode U so that they only contain $\uparrow^{\mathsf{U}}_{\mathsf{L}} A_{\mathsf{L}}$. In this version we have $!A = \downarrow^{\mathsf{U}}_{\mathsf{L}} \uparrow^{\mathsf{U}}_{\mathsf{L}} A$. Again, the rules of dual intuitionistic linear logic are modeled precisely.

▶ **Example 5** (Intuitionistic Subexponential Linear Logic). Subexponential linear logic [40, 41] also uses a preorder of modes, each of which permits specific structural rules. We obtain a formulation of *intuitionistic subexponential linear logic* by adding a new distinguished mode L with $m \geq \mathsf{L}$ for all given subexponential modes m, retaining all the other relations. We further restrict all modes m except for L to contain only $\uparrow^{m}_{\mathsf{L}} A_{\mathsf{L}}$, forcing all logical inferences to take place at mode L.

Compared to [16] our system does not contain $?A$ and is not focused; compared to [29], our base logic is linear rather than ordered. Also, all of our structural rules are implicit.

3 Adjoint Natural Deduction

Substructural *sequent calculi* have recently found interesting computational interpretations [12, 54, 13, 43, 48], including adjoint logic [47]. In this paper, we look instead at *functional* interpretations, which are most closely related to *natural deduction*. Some guide is provided by natural deduction systems for *linear logic* (see, for example, [2, 10, 52]), but already they are not entirely straightforward. For example, some of these calculi do not satisfy subject reduction. The interplay between modes and substructural properties creates some further

complications. The closest blueprint to follow is probably Benton's [8], but his system does not exhibit the full generality of adjoint logic and is also not quite "parsimonious" in the sense of the LNL sequent calculus.

In the interest of economy, we present the calculus with proof terms and two bidirectional typing judgments, $\Delta \vdash e \Longleftarrow A_m$ (expression e checks against A_m) and $\Delta \vdash s \Longrightarrow A_m$ (expression s *synthesizes* A_m). The syntax for expressions is summarized in Figure 2. The bidirectional nature will allow us to establish a precise relationship to the sequent calculus (Section 4), but it does not immediately yield a type checking algorithm since the context merge operation is highly nondeterministic when used to split contexts. An algorithmic system can be found in Section 6.

We obtain the vanilla typing judgment by replacing both checking and synthesis judgments with $\Delta \vdash e : A$, dropping the rules $\Rightarrow/\Leftarrow$ and $\Leftarrow/\Rightarrow$, and removing the syntactic form $(e : A_m)$. We further obtain a pure natural deduction system by removing the proof terms, although uses of the hypothesis rule then need to be annotated with variables in order to avoid any ambiguities.

Checkable Exps.

e	::=	$\lambda x.\, e$	$(\multimap)$
	$\mid$	$\{\ell \Rightarrow e_\ell\}_{\ell \in L}$	$(\&)$
	$\mid$	**susp** e	$(\uparrow)$
	$\mid$	(e_1, e_2)	$(\otimes)$
	$\mid$	$(\,)$	$(\mathbf{1})$
	$\mid$	$\ell(e)$	$(\oplus)$
	$\mid$	**down** e	$(\downarrow)$
	$\mid$	**match** s M	
	$\mid$	s	

Synthesizable Exps.

s	::=	x	
	$\mid$	$s\ e$	$(\multimap)$
	$\mid$	$s.\ell$	$(\&)$
	$\mid$	**force** s	$(\uparrow)$
	$\mid$	$(e : A_m)$	

Matches

M	::=	$(x_1, x_2) \Rightarrow e'$	$(\otimes)$
	$\mid$	$(\,) \Rightarrow e'$	$(\mathbf{1})$
	$\mid$	$(\ell(x) \Rightarrow e_\ell)_{\ell \in L}$	$(\oplus)$
	$\mid$	**down** $x \Rightarrow e'$	$(\downarrow)$

Figure 2 Expressions for Bidirectional Natural Deduction.

The rules maintain a few important invariants, particularly *independence*:

(i) $\Delta \vdash e \Longleftarrow A_m$ presupposes $\Delta \geq m$ **(ii)** $\Delta \vdash s \Longrightarrow A_m$ presupposes $\Delta \geq m$

This is somewhat surprising because we think of the synthesis judgment $s \Longrightarrow A_m$ as proceeding top-down rather than bottom-up. Indeed, there are other choices with dependence and structural properties being checked in different places. We picked this particular form because we want general typing $e : A_m$ to arise from collapsing the checking/synthesis distinction. This means that the two rules $\Rightarrow/\Leftarrow$ and $\Leftarrow/\Rightarrow$ should have no conditions because those would disappear. The algorithmic system in Section 6 checks the conditions in different places.

As an example of interesting rules we revisit $\downarrow_m^n A_n$ (where $n \geq m$ is presupposed). The introduction rule of natural deduction mirrors the right rule of the sequent calculus, which is the case throughout.

$$\frac{\Gamma' \geq n \quad \Gamma' \vdash A_n}{\Gamma_{\mathsf{W}}\,;\Gamma' \vdash \downarrow_m^n A_n}\ \downarrow R \qquad \frac{\Delta' \geq n \quad \Delta' \vdash e \Longleftarrow A_n}{\Delta_{\mathsf{W}}\,;\Delta' \vdash \mathbf{down}\ e \Longleftarrow \downarrow_m^n A_n}\ \downarrow I$$

As is typical for these translations, the elimination rules turns the left rule "upside down" because (like all rules in natural deduction) the principal formula is on the right-hand side of judgment, not the left as in the sequent calculus. This means we now have some conditions to check.

$$\frac{\Delta \vdash s \Longrightarrow A_m}{\Delta \vdash s \Longleftarrow A_m}\ \Rightarrow/\Leftarrow \qquad \frac{\Delta \vdash e \Longleftarrow A_m}{\Delta \vdash (e : A_m) \Longrightarrow A_m}\ \Leftarrow/\Rightarrow \qquad \frac{}{\Delta_{\mathsf{W}} ; x : A_m \vdash x \Longrightarrow A_m}\ \mathsf{hyp}$$

$$\frac{\Delta, x : A_m \vdash e \Longleftarrow B_m}{\Delta \vdash \lambda x.\, e \Longleftarrow A_m \multimap B_m}\ \multimap I \qquad \frac{\Delta \vdash s \Longrightarrow A_m \multimap B_m \quad \Delta' \vdash e \Longleftarrow A_m}{\Delta\,;\Delta' \vdash s\, e \Longrightarrow B_m}\ \multimap E$$

$$\frac{\Delta \vdash e_\ell \Longleftarrow A^\ell_m \quad (\forall \ell \in L)}{\Delta \vdash \{\ell \Rightarrow e_\ell\}_{\ell\in L} \Longleftarrow \&\{\ell : A^\ell_m\}_{\ell\in L}}\ \&I \qquad \frac{\Delta \vdash s \Longrightarrow \&\{\ell : A^\ell_m\}_{\ell\in L} \quad (\ell \in L)}{\Delta \vdash s.\ell \Longrightarrow A^\ell_m}\ \&E$$

$$\frac{\Delta \vdash e \Longleftarrow A_k}{\Delta \vdash \mathbf{susp}\ e \Longleftarrow \uparrow^m_k A_k}\ \uparrow I \qquad \frac{\Delta' \geq m \quad \Delta' \vdash s \Longrightarrow \uparrow^m_k A_k}{\Delta_{\mathsf{W}}\,;\Delta' \vdash \mathbf{force}\ s \Longrightarrow A_k}\ \uparrow E$$

$$\frac{\Delta \vdash e_1 \Longleftarrow A_m \quad \Delta' \vdash e_2 \Longleftarrow B_m}{\Delta\,;\Delta' \vdash (e_1, e_2) \Longleftarrow A_m \otimes B_m}\ \otimes I$$

$$\frac{\Delta \vdash s \Longrightarrow A_m \otimes B_m \quad \Delta \geq m \geq r \quad \Delta', x_1 : A_m, x_2 : B_m \vdash e' \Longleftarrow C_r}{\Delta\,;\Delta' \vdash \mathbf{match}\ s\ ((x_1, x_2) \Rightarrow e') \Longleftarrow C_r}\ \otimes E$$

$$\frac{}{\Delta_{\mathsf{W}} \vdash (\,) \Longleftarrow \mathbf{1}_m}\ \mathbf{1}I \qquad \frac{\Delta \vdash s \Longrightarrow \mathbf{1}_m \quad \Delta \geq m \geq r \quad \Delta' \vdash e' \Longleftarrow C_r}{\Delta\,;\Delta' \vdash \mathbf{match}\ s\ (() \Rightarrow e') \Longleftarrow C_r}\ \mathbf{1}E$$

$$\frac{\Delta \vdash e \Longleftarrow A^\ell_m}{\Delta \vdash \ell(e) \Longleftarrow \oplus\{\ell : A^\ell_m\}_{\ell\in L}}\ \oplus I$$

$$\frac{\Delta \vdash s \Longrightarrow \oplus\{\ell : A^\ell_m\}_{\ell\in L} \quad \Delta \geq m \geq r \quad \Delta', x : A^\ell_m \vdash e_\ell \Longleftarrow C_r \quad (\forall \ell \in L)}{\Delta\,;\Delta' \vdash \mathbf{match}\ s\ (\ell(x) \Rightarrow e_\ell)_{\ell\in L} \Longleftarrow C_r}\ \oplus E$$

$$\frac{\Delta' \geq n \quad \Delta' \vdash e \Longleftarrow A_n}{\Delta_{\mathsf{W}}\,;\Delta' \vdash \mathbf{down}\ e \Longleftarrow \downarrow^n_m A_n}\ \downarrow I \qquad \frac{\Delta \vdash s \Longrightarrow \downarrow^n_m A_n \quad \Delta \geq m \geq r \quad \Delta', x : A_n \vdash e' \Longleftarrow C_r}{\Delta\,;\Delta' \vdash \mathbf{match}\ s\ (\mathbf{down}\ x \Rightarrow e') \Longleftarrow C_r}\ \downarrow E$$

Figure 3 Implicit Bidirectional Natural Deduction.

$$\frac{\Gamma, A_n \vdash C_r}{\Gamma\,; \downarrow^n_m A_n \vdash C_r}\ \downarrow L \qquad \frac{\Delta \vdash s \Longrightarrow \downarrow^n_m A_n \quad \Delta \geq m \geq r \quad \Delta', x : A_n \vdash e' \Longleftarrow C_r}{\Delta\,;\Delta' \vdash \mathbf{match}\ s\ (\mathbf{down}\ x \Rightarrow e') \Longleftarrow C_r}\ \downarrow E$$

$\Delta \geq m$ is needed to enforce independence on the first premise. $m \geq r$ together with $n \geq m$ enforces independence on the second premise. Similar restrictions appear in the other elimination rules for the positive connectives ($\otimes$, $\mathbf{1}$, $\oplus$).

We often say a natural deduction is *normal*, which means that it cannot be reduced, but under which collection of reductions? The difficulty here is that rewrite rules that reduce an introduction of a connective immediately followed by its elimination are not sufficient to achieve deductions that are *analytic* in the sense that they satisfy the subformula property. To obtain analytic deductions, we have to add *permuting conversions*.

We follow a different approach by directly characterizing *verifications* [21, 36], which are proofs that can be seen as constructed by applying introduction rules bottom-up and elimination rules top-down. By definition, verifications satisfy the subformula property and are therefore analytic and a suitable "normal form" even without defining a set of reductions.

How does this play out here? It turns out that if $\Delta \vdash e \Longleftarrow A_m$ then the corresponding proof of A_m (obtained by erasure of expressions) is a *verification* if the $\Leftarrow/\Rightarrow$ rule is disallowed and the $\Rightarrow/\Leftarrow$ rule is restricted to atomic propositions P. As we will see in Section 4, this corresponds precisely to a cut-free sequent calculus derivation where the identity is restricted to atomic propositions. Proof-theoretically, the meaning of a proposition is determined by its *verifications*, which, by definition, only decompose the given proposition into its components. Compare this with general proofs that do not obey such a restriction.

In the next section we will prove that every proposition that has a natural deduction also has a verification by relating the sequent calculus and natural deduction.

▶ **Example 6** (Church's λI calculus)**.** Church [18] introduced the λI calculus in which each bound variable requires at least one occurrence. We obtain the simply-typed λI calculus with one mode S with $\sigma(\mathsf{S}) = \{\mathsf{C}\}$ and using $A_\mathsf{s} \multimap B_\mathsf{s}$ as the only type constructor.

Similarly, we obtain the simply-typed λ-calculus with a single mode U with $\sigma(\mathsf{U}) = \{\mathsf{W}, \mathsf{C}\}$ and the simply-typed *linear* λ-calculus with a single mode L with $\sigma(\mathsf{L}) = \{\,\}$, using $A_\mathsf{L} \multimap B_\mathsf{L}$ as the only type constructor.

▶ **Example 7** (Intuitionistic Natural Deduction)**.** We obtain (structural) intutionistic natural deduction with a single mode U with $\sigma(\mathsf{U}) = \{\mathsf{W}, \mathsf{C}\}$, where we can define $A \vee B = \oplus\{\mathbf{inl} : A, \mathbf{inr} : B\}$ and $\bot = \oplus\{\,\}$, $A \wedge B = \&\{\pi_1 : A, \pi_2 : B\}$ and $\top = \&\{\,\}$ and $A \to B = A \multimap B$.

▶ **Example 8** (Intuitionistic S4)**.** We obtain the fragment of intuitionistic S4 in its dual formulation [42] without possibility ($\Diamond A$) with two modes V and U with $\mathsf{V} > \mathsf{U}$ and $\sigma(\mathsf{V}) = \sigma(\mathsf{U}) = \{\mathsf{W}, \mathsf{C}\}$. As in the DILL example of the adjoint sequent calculus, the mode V is inhabited only by types $\uparrow^\mathsf{V}_\mathsf{U} A_\mathsf{U}$ and we define $\Box A_\mathsf{U} = \downarrow^\mathsf{V}_\mathsf{U} \uparrow^\mathsf{V}_\mathsf{U} A_\mathsf{U}$, which is a *comonad*. The judgment $\Delta \,;\, \Gamma \vdash C$ *true* with valid hypotheses Δ and true hypothesis Γ is modeled by $\Delta_\mathsf{V}, \Gamma_\mathsf{U} \vdash C_\mathsf{U}$.

The structure of *verifications* is modeled almost exactly with one small exception: we allow the form $\Delta_\mathsf{V} \vdash C_\mathsf{V}$. Because any proposition $B_\mathsf{V} = \uparrow^\mathsf{V}_\mathsf{U} A_\mathsf{U}$, there is only one applicable rule to construct a verification of this judgment: $\uparrow I$ (which, not coincidentally, is invertible).

▶ **Example 9** (Lax Logic)**.** We obtain natural deduction for lax logic [9, 42] with two modes, U and X, with $\mathsf{U} > \mathsf{X}$ and $\sigma(\mathsf{U}) = \sigma(\mathsf{X}) = \{\mathsf{W}, \mathsf{C}\}$. The mode X is inhabited only by $\downarrow^\mathsf{U}_\mathsf{X} A_\mathsf{U}$. We define $\bigcirc A_\mathsf{U} = \uparrow^\mathsf{U}_\mathsf{X} \downarrow^\mathsf{U}_\mathsf{X} A_\mathsf{U}$, which is a strong monad [9].

We model the rules of Pfenning and Davies [42] exactly, except that we allow hypotheses B_X, which must have the form $\downarrow^\mathsf{U}_\mathsf{X} A_\mathsf{U}$. We can eagerly apply $\downarrow E$ to obtain A_U, which again does not lose completeness by the invertibility of $\downarrow L$ in the sequent calculus. We can also obtain linear versions of these relationships following [11], although the term calculi do not match up exactly.

4 Relating Sequent Calculus and Natural Deduction

Rather than trying to find a complete set of proof reductions for natural deduction, we translate a proof to the sequent calculus, apply cut and identity elimination, and then translate the resulting proof back to natural deduction. This is not essential, but it simultaneously proves the soundness and completeness of natural deduction for adjoint logic and the completeness of verifications. This allows us to focus on the computational interpretation in Section 5 that is a form of substructural functional programming.

In general we see the following patterns in the correctness proofs:

- The identity corresponds to $\Rightarrow/\Leftarrow$
- Cut corresponds to $\Leftarrow/\Rightarrow$
- Right rules correspond to introduction rules
- Left rules correspond to upside-down elimination rules
  - For negative connectives ($\multimap$, $\&$, $\uparrow$) they are just reversed
  - For positive connectives ($\otimes$, $\mathbf{1}$, $\oplus$, $\downarrow$) in addition a new hypothesis is introduced in a second premise

The last point justifies reading a hypothesis $x : A_m$ as $x \Longrightarrow A_m$.

For completeness of natural deduction, one might expect to prove that $\Gamma \vdash C$ in the sequent calculus implies $\Gamma \vdash e \Longleftarrow C$ in natural deduction. While this holds, a direct proof would not generate a verification from a cut-free proof. Intuitively, the way the proof proceeds instead is to take a sequent $x_1 : A_1, \ldots, x_n : A_n \vdash C$ (ignoring modes for the moment) and annotate each antecedent with a synthesizing term and the succedent with an expression $s_1 \Longrightarrow A_1, \ldots, s_n \Longrightarrow A_n \vdash e \Longleftarrow C$. This means we have to account for the variables in s_i, and we do this with a substitution θ assigning synthesizing terms to each antecedent in Γ. We therefore define substitutions as mapping from variables to synthesizing terms.

$$\text{Substitutions} \quad \theta \quad ::= \quad \cdot \mid \theta, x \mapsto s$$

We type substitutions with the judgment $\Delta \vdash \theta \Longrightarrow \Gamma$, where Δ contains the free variables in θ. This judgment must respect independence and the structural properties of each antecedent in Γ, as defined by the following rules:

$$\frac{}{\cdot \vdash (\cdot) \Longrightarrow (\cdot)} \qquad \frac{\Delta \vdash \theta \Longrightarrow \Gamma \quad \Delta' \geq m \quad \Delta' \vdash s \Longrightarrow A_m}{\Delta\,;\Delta' \vdash (\theta, x \mapsto s) \Longrightarrow (\Gamma, x : A_m)}$$

We will use silently that if $\Delta \vdash \theta \Longrightarrow \Gamma$ and $\Gamma \geq m$ then $\Delta \geq m$.

We write $e(x)$ and $s'(x)$ for terms with (possibly multiple, possibly no) occurrences of x and $e(s)$ and $s'(s)$ for the result of substituting s for x, respectively. Because variables $x : A$ synthesize their types $x \Longrightarrow A$, the following admissible rules are straightforward assuming the premises satisfy our presuppositions.

▶ **Theorem 10** (Substitution Property). *The following properties are admissible:*

(i) *If* $\Delta \vdash s \Longrightarrow A_m$ *and* $\Delta', x : A_m \vdash e(x) \Longleftarrow C_r$ *then* $\Delta\,;\Delta' \vdash e(s) \Longleftarrow C_r$

(ii) *If* $\Delta \vdash s \Longrightarrow A_m$ *and* $\Delta', x : A_m \vdash s'(x) \Longrightarrow B_k$ *then* $\Delta\,;\Delta' \vdash s'(s) \Longrightarrow B_k$

Proof. By a straightforward simultaneous rule induction on the second given derivation. In some cases we need to apply monotonicity. For example, if m admits contraction and $\Delta \geq m$, then each hypothesis in Δ must also admit contraction. ◀

Now we have the pieces in place to prove the translation from the sequent calculus to natural deduction.

▶ **Lemma 11** (Context Split). *If* $\Delta \vdash \theta \Longrightarrow (\Gamma;\Gamma')$ *then there exists* θ_1 *and* θ_2 *and* Δ_1 *and* Δ_2 *such that* $\Delta = \Delta_1;\Delta_2$ *and* $\Delta_1 \vdash \theta_1 \Longrightarrow \Gamma$ *and* $\Delta_2 \vdash \theta_2 \Longrightarrow \Gamma'$.

Proof. By case analysis on the definition of context merge operation and induction on $\Delta \vdash \theta \Longrightarrow (\Gamma;\Gamma')$. We rely on associativity and commutativity of context merge. We show two cases.

Case: $(\Gamma_1, x : A_m)\,;(\Gamma_2, x : A_m) = (\Gamma_1\,;\Gamma_2), x : A_m$ and $\mathsf{C} \in \sigma(m)$

$$\frac{\Delta \vdash \theta_{12} \Longrightarrow \Gamma_1\,;\Gamma_2 \quad \Delta' \geq k \quad \Delta' \vdash s \Longrightarrow A_m}{\Delta\,;\Delta' \vdash (\theta_{12}, x \mapsto s) \Longrightarrow (\Gamma_1\,;\Gamma_2), x : A_m}$$

$\Delta_1 \vdash \theta_1 \Longrightarrow \Gamma_1$ and	
$\Delta_2 \vdash \theta_2 \Longrightarrow \Gamma_2$ and	
$\Delta = \Delta_1\,;\Delta_2$	by IH
$\Delta_1\,;\Delta' \vdash \theta_1, x \mapsto s \Longrightarrow \Gamma_1, x : A_m$	by rule
$\Delta_2\,;\Delta' \vdash \theta_2, x \mapsto s \Longrightarrow \Gamma_2, x : A_m$	by rule
since $\mathsf{C} \in \sigma(m)$ and $\Delta' \geq m$, we have $\mathsf{C} \in \sigma(k)$ for any $B_k \in \Delta'$	by monotonicity
$(\Delta_1\,;\Delta')\,;(\Delta_2\,;\Delta') = (\Delta_1\,;\Delta_2)\,;\Delta' = \Delta\,;\Delta'$	by previous line

Case: $\Gamma_1\,;(\Gamma_2, x : A_m) = (\Gamma_1\,;\Gamma_2), x : A_m$ and $x \notin \mathrm{dom}(\Gamma_1)$

$$\frac{\Delta \vdash \theta_{12} \Longrightarrow \Gamma_1\,;\Gamma_2 \quad \Delta' \geq k \quad \Delta' \vdash s \Longrightarrow A_m}{\Delta\,;\Delta' \vdash (\theta_{12}, x \mapsto s) \Longrightarrow ((\Gamma_1\,;\Gamma_2), x \mapsto A_m)}$$

$\Delta_1 \vdash \theta_1 \Longrightarrow \Gamma_1$ and	
$\Delta_2 \vdash \theta_2 \Longrightarrow \Gamma_2$ and	
$\Delta = \Delta_1\,;\Delta_2$	by IH
$\Delta_2\,;\Delta' \vdash \theta_2, x \mapsto s \Longrightarrow \Gamma_2, x \mapsto A_m$	by rule
$\Delta_1\,;(\Delta_2\,;\Delta') = (\Delta_1\,;\Delta_2)\,;\Delta' = \Delta\,;\Delta'$	by associativity of context merge

◀

▶ **Theorem 12** (From Sequent Calculus to Natural Deduction). *theoremseqtond*
If $\Gamma \vdash A_r$ *and* $\Delta \vdash \theta \Longrightarrow \Gamma$ *then* $\Delta \vdash e \Longleftarrow A_r$ *for some* e.

Proof. By rule induction on the derivation $\mathcal{D}$ of $\Gamma \vdash A_r$ and applications of inversion on the definition of substitution. We present several indicative cases. In this proof we write out the variables labeling the antecedents in sequents to avoid ambiguities.

Case: $\mathcal{D}$ ends in the identity.

$$\mathcal{D} = \quad \frac{}{\Gamma_{\mathsf{W}}\,; x : A_m \vdash A_m}\ \mathsf{id}$$

$\Delta \vdash \theta \Longrightarrow (\Gamma_{\mathsf{W}}\,; x : A_m)$	Given
$\theta = (\theta_{\mathsf{W}}, x \mapsto s)$	By inversion
$\Delta = (\Delta_{\mathsf{W}}\,;\Delta')$ with $\Delta_W \vdash \theta_{\mathsf{W}} \Longrightarrow \Gamma_{\mathsf{W}}$ and $\Delta' \vdash s \Longrightarrow A_m$	By context split
Δ_{W} satisfies weakening	By monotonicity
$\Delta' \vdash s \Longleftarrow A_m$	By rule $\Rightarrow/\Leftarrow$
$\Delta_{\mathsf{W}}\,;\Delta' \vdash s \Longleftarrow A_m$	By weakening
$\Delta \vdash s \Longleftarrow A_m$	Since $\Delta = (\Delta_{\mathsf{W}}\,;\Delta')$

Case: $\mathcal{D}$ ends in cut.

$$\mathcal{D} = \quad \frac{\Gamma_1 \geq m \geq r \quad \overset{\mathcal{D}_1}{\Gamma_1 \vdash A_m} \quad \overset{\mathcal{D}_2}{\Gamma_2, x : A_m \vdash C_r}}{\Gamma_1\,;\Gamma_2 \vdash C_r}\ \mathsf{cut}$$

$\Delta \vdash \theta \Longrightarrow (\Gamma_1 \,;\, \Gamma_2)$ Given
$\Delta = (\Delta_1 \,;\, \Delta_2)$, $\theta = (\theta_1, \theta_2)$ with $\Delta_1 \vdash \theta_1 \Longrightarrow \Gamma_1$ and $\Delta_2 \vdash \theta_2 \Longrightarrow \Gamma_2$ By context split
$\Delta_1 \vdash e_1 \Longleftarrow A_m$ By IH on $\mathcal{D}_1$
$\Delta_1 \vdash (e_1 : A_m) \Longrightarrow A_m$ by rule $\Leftarrow/\Rightarrow$
$\Delta_2, x : A_m \vdash (\theta_2, x \mapsto x) \Longrightarrow (\Gamma_2, x : A_m)$ By subst. rule
$\Delta_2, x : A_m \vdash e_2(x) \Longleftarrow C_r$ By IH on $\mathcal{D}_2$
$\Delta_1 \,;\, \Delta_2 \vdash e_2(e_1 : A_m) \Longleftarrow C_r$ By substitution (Theorem 10)
◀

While there are no substitutions involved, the other direction has to take care to introduce a cut *only* for uses of the $\Leftarrow/\Rightarrow$ rule, and identity *only* for uses of the $\Rightarrow/\Leftarrow$ rule. This requires a generalization of the induction hypothesis so that the elimination rules can be turned "upside down".

▶ **Theorem 13** (From Natural Deduction to Sequent Calculus). *theoremndtoseq*
(i) *If* $\Delta \vdash e \Longleftarrow C_r$ *then* $\Delta \vdash C_r$
(ii) *If* $\Delta \vdash s \Longrightarrow A_m$ *and* $\Delta', x : A_m \vdash C_r$ *then* $\Delta \,;\, \Delta' \vdash C_r$

Proof. By simultaneous rule induction on $\Delta \vdash e \Longleftarrow C_r$ and $\Delta \vdash s \Longrightarrow A_m$. We provide several sample cases.
Case: The derivation ends in $\Rightarrow/\Leftarrow$.

$$\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{D}' \\ \Delta \vdash s \Longrightarrow A_m\end{array}}{\Delta \vdash s \Longleftarrow A_m}\ \Rightarrow/\Leftarrow$$

$x : A_m \vdash A_m$ By identity rule
$\Delta \vdash A_m$ By IH(ii) with $\Delta' = (\cdot)$
Case: The derivation ends in $\Leftarrow/\Rightarrow$.

$$\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{D}' \\ \Delta \vdash e \Longleftarrow A_m\end{array}}{\Delta \vdash (e : A_m) \Longrightarrow A_m}\ \Leftarrow/\Rightarrow$$

$\Delta', x : A_m \vdash C_r$ and $\Delta \geq r$ Assumption
$\Delta \vdash A_m$ By IH(i) on $\mathcal{D}'$
$\Delta \,;\, \Delta' \vdash C_m$ By rule of cut
◀

As mentioned above, verifications are the foundational equivalent of normal forms in natural deduction. Using the two translations above we can show that every provable proposition has a verification. While we have not written the translations out as functions, they constitute the computational contents of our constructive proof of Theorem 12 and Theorem 13.

▶ **Theorem 14.** *If* $\Delta \vdash e \Longleftarrow A_m$ *then there exists a verification of* $\Delta \vdash e \Longleftarrow A_m$.

Proof. Given an arbitrary deduction of $\Delta \vdash e \Longleftarrow A_m$, we can use Theorem 13 (i) to translate it to a sequent derivation of $\Delta \vdash A_m$. By the admissibility of cut and identity (Theorem 2), we can obtain a long cut-free proof of $\Delta \vdash A_m$. We observe that the translation of Theorem 12 translates only cut to $\Leftarrow/\Rightarrow$ and only identity to $\Rightarrow/\Leftarrow$. Using the translation back to natural deduction from a long cut-free proof therefore results in a verification. ◀

5 Dynamics

As mentioned in Section 3, we obtain a simple typing judgment $\Delta \vdash e : A$ by collapsing the distinction between $e \Longleftarrow A$ and $s \Longrightarrow A$, using e as a universal notation for all expressions. Furthermore, the annotation $(e : A_m)$ is removed and the rules $\Rightarrow/\Leftarrow$ and $\Leftarrow/\Rightarrow$ are also removed. The resulting rules remain *syntax-directed* in the sense that for every form of expression there is a unique typing rule.

We further annotate the mode-changing constructors with the mode of their subject, which in each case is uniquely determined by the typing derivation. Some of these annotations are necessary, because the computation rules depend on them; other information is redundant but kept for clarity.

- $\mathbf{susp}^m_k\ e : \uparrow^m_k A_k$ if $e : A_k$
- $\mathbf{down}^n_m\ e : \downarrow^n_m A_n$ if $e : A_n$
- $\mathbf{force}^m_k\ e : A_k$ if $e : \uparrow^m_k A_k$
- $\mathbf{match}_m\ e\ M_r : C_r$ if $e : A_m$

We give a sequential call-by-value semantics similar to the K machine (e.g., [26]), but maintaining a global environment similar to the Milner Abstract Machine [3]. There are two forms of state in the machine:

- $\eta\ ;\ K \rhd_m e$ (evaluate e of mode m under continuation stack K and environment η)
- $\eta\ ;\ K \blacktriangleleft_m v$ (pass value v of mode m to continuation stack K in environment η)

In the first, e is an expression to be evaluated and K is a stack of continuations that the value of e is passed to for further computation. The second then passes this value v to the continuation stack.

The global environment η maps variables to values, but these values may again reference other variables. In this way it is like Launchbury's [32] heap. We can exploit this to model the call-by-need evaluation strategy, which can be found in our extended version (link in preamble). Because we maintain a global environment, we do not need to build closures, nor do we need to substitute values for variables. Instead, we only (implicitly) rename variables to make them globally unique. This form of specification allows us to isolate the *dynamic use of variables*, which means we can observe the computational consequences of modes and their substructural nature. We could also use the translation to the sequent calculus and then observe the consequence with an explicit heap [48, 43], but in this paper we study natural deduction and functional computation more directly.

The syntax for continuations, environments, values, and machine states is summarized in Figure 4. Although not explicitly polarized (as in [33]), values of negative type ($\multimap$, $\&$, $\uparrow$) are lazy in the sense that they abstract over unevaluated expressions, while values of positive types ($\otimes$, $\mathbf{1}$, $\oplus$, $\downarrow$) are constructed from other values. This will be significant in our analysis of the computational properties of modes. Continuation frames just reflect the left-to-right call-by-value nature of evaluation.

Values are typed as expressions. Frames are typed with $\Gamma \vdash f : B_k < A_m$, which means f takes a value of type A_m and passes a value of type B_k further up the continuation stack. We show the rules for continuations. Note that the non-empty continuation rule has a mode k in the premises that doesn't appear in the conclusion.

$$\frac{}{\Delta_{\mathsf{W}} \vdash \epsilon : A_m < A_m} \qquad \frac{\Delta \vdash K : C_r < B_k \quad \Delta' \vdash f : B_k < A_m}{\Delta\ ;\ \Delta' \vdash K \cdot f : C_r < A_m}$$

Regarding environments we face a fundamental choice. One possibility is to extend the term language of natural deduction with explicit constructs for weakening and contraction. Then, similar to Girard and Lafont [24], no garbage collection would be required during evaluation since uniqueness of references to variables would be maintained.

Frames

$$
\begin{array}{lcll}
f & ::= & \_\ e_2 \mid v_1\ \_ & (\multimap) \\
& \mid & \_.\ell & (\&) \\
& \mid & \mathbf{force}_k^m\ \_ & (\uparrow) \\
& \mid & (\_, e_2) \mid (v_1, \_) & (\otimes) \\
& \mid & \ell(\_) & (\oplus) \\
& \mid & \mathbf{down}_m^n\ \_ & (\downarrow) \\
& \mid & \mathbf{match}_m\ \_\ M_r & (\otimes, \mathbf{1}, \oplus, \downarrow)
\end{array}
$$

Continuations

$$K ::= \epsilon \mid K \cdot f$$

Environments

$$\eta ::= \cdot \mid \eta, x \mapsto v \mid \eta, [x \mapsto v]$$

States

$$
\begin{array}{lcl}
S & ::= & \eta\ ;\ K \rhd_m e \\
& \mid & \eta\ ;\ K \blacktriangleleft_m v
\end{array}
$$

Values

$$
\begin{array}{lcll}
v & ::= & \lambda x.\, e(x) & (\multimap) \\
& \mid & \{\ell \Rightarrow e_\ell\}_{\ell \in L} & (\&) \\
& \mid & \mathbf{susp}_k^m\ e & (\uparrow) \\
& \mid & (v_1, v_2) & (\otimes) \\
& \mid & () & (\mathbf{1}) \\
& \mid & \ell(v) & (\oplus) \\
& \mid & \mathbf{down}_m^n\ v & (\downarrow)
\end{array}
$$

Figure 4 Machine States.

We pursue here an alternative that leads to slightly deeper properties. We leave the structural rules implicit as in the rules so far. This means that variables of linear mode (that is, a mode that allows neither weakening nor contraction) have uniqueness of reference and their bindings can be deallocated when dereferenced. Variables of structural mode (that is, a mode that allows both weakening and contraction) are simply persistent in the dynamics and therefore could be subject to an explicit garbage collection algorithm.

A difficulty arises with variables that only admit contraction but not weakening. After they are dereferenced the first time, they may or may not be dereferenced again. That is, they could be implicitly weakened after the first access. In order to capture this we introduce a new form of typing $[x : A_m]$ and binding $[x \mapsto v]$ we call *provisional*. A provisional binding does not need to be referenced even if m does not admit weakening. The important new property is that an "ordinary" variable $y : A_k$ that does not admit weakening can not appear in a binding $[x \mapsto v]$. In addition, all the usual independence requirements have to be observed.

The rules for typing expressions, continuations, etc. are extended in the obvious way, allowing variables $[x : A_m]$ to be used or ignored (as a part of some Δ_W). We extend the context merge operation as follows, keeping in mind that $x : A_m$ may require an occurrence of x (depending on $\sigma(m)$), while $[x : A_m]$ does not.

$$
\begin{array}{rcrcll}
(\Delta_1, [x : A_m]) & ; & (\Delta_2, [x : A_m]) & = & (\Delta_1\ ;\ \Delta_2), [x : A_m] & \text{provided } \mathsf{C} \in \sigma(m) \\
(\Delta_1, x : A_m) & ; & (\Delta_2, [x : A_m]) & = & (\Delta_1\ ;\ \Delta_2), x : A_m & \text{provided } \mathsf{C} \in \sigma(m) \\
(\Delta_1, [x : A_m] & ; & (\Delta_2, x : A_m) & = & (\Delta_1\ ;\ \Delta_2), x : A_m & \text{provided } \mathsf{C} \in \sigma(m) \\
(\Delta_1, [x : A_m]) & ; & \Delta_2 & = & (\Delta_1\ ;\ \Delta_2), [x : A_m] & \text{provided } x \notin \mathrm{dom}(\Delta_2) \\
\Delta_1 & ; & (\Delta_2, [x : A_m]) & = & (\Delta_1\ ;\ \Delta_2), [x : A_m] & \text{provided } x \notin \mathrm{dom}(\Delta_1)
\end{array}
$$

We have the following typing rules for environments. Δ_W now means that every declaration in Δ can be weakened, either explicitly because its mode allows weakening, or implicitly because it is provisional.

$$
\frac{}{(\cdot) : (\cdot)} \qquad \frac{\eta : (\Delta\ ;\ \Delta') \quad \Delta' \geq m \quad \Delta' \vdash v : A_m}{(\eta, x \mapsto v) : (\Delta, x : A_m)} \qquad \frac{\eta : (\Delta\ ;\ \Delta'_{\mathsf{W}}) \quad \Delta'_{\mathsf{W}} \geq m \quad \Delta'_{\mathsf{W}} \vdash v : A_m}{(\eta, [x \mapsto v]) : (\Delta, [x : A_m])}
$$

As an example, consider $\eta_0 = (x \mapsto (\,), y \mapsto \lambda f.\, f\, x)$ where the mode of variables is immaterial, but let's fix them to be L with $\sigma(\mathsf{L}) = \{\,\}$.

$$\dfrac{\dfrac{\dfrac{}{(\cdot) : (\cdot)} \quad \dfrac{}{\cdot \vdash (\,) : \mathbf{1}_{\mathsf{L}}}}{(x \mapsto (\,)) : (x : \mathbf{1}_{\mathsf{L}})} \quad \begin{array}{c}\vdots\\ x : \mathbf{1}_{\mathsf{L}} \vdash \lambda f.\, f\, x : (\mathbf{1}_{\mathsf{L}} \multimap A_{\mathsf{L}}) \multimap A_{\mathsf{L}}\end{array}}{(x \mapsto (\,), y \mapsto \lambda f.\, f\, x) : (y : (\mathbf{1}_{\mathsf{L}} \multimap A_{\mathsf{L}}) \multimap A_{\mathsf{L}})}$$

We observe that the binding of $x \mapsto (\,)$ does not contribute a declaration $x : \mathbf{1}$ to the result context due to the occurrence of x in the value of y.

Now consider a slightly modified version where the mode of both x and y is S with $\sigma(\mathsf{S}) = \{\mathsf{C}\}$, and the binding of $y \mapsto \ldots$ becomes provisional. This modified example is no longer well-typed.

$$\dfrac{\dfrac{\dfrac{}{(\cdot) : (\cdot)} \quad \dfrac{}{\cdot \vdash (\,) : \mathbf{1}_{\mathsf{S}}}}{(x \mapsto (\,)) : (x : \mathbf{1}_{\mathsf{S}})} \quad \begin{array}{c}\vdots\\ x : \mathbf{1}_{\mathsf{S}} \vdash \lambda f.\, f\, x : (\mathbf{1}_{\mathsf{S}} \multimap A_{\mathsf{S}}) \multimap A_{\mathsf{S}}\end{array}}{(x \mapsto (\,), [y \mapsto \lambda f.\, f\, x]) : (y : [(\mathbf{1}_{\mathsf{S}} \multimap A_{\mathsf{S}}) \multimap A_{\mathsf{S}})]}\ ??$$

The problem is at the rule application marked by ??. The variable y does not need to be used, despite its mode, because the binding is provisional. This means that x might also not be used because its only occurrence is in the value of y. But that is not legal, since the mode of x does not admit weakening and the binding is not provisional.

We type abstract machine states with the type of their final answer, that is $S : C_r$.

$$\dfrac{\eta : (\Delta\,;\Delta') \quad \Delta \vdash K : C_r < A_m \quad \Delta' \vdash e : A_m}{(\eta\,;\, K \rhd_m e) : C_r} \qquad \dfrac{\eta : (\Delta\,;\Delta') \quad \Delta \vdash K : C_r < A_m \quad \Delta' \vdash v : A_m}{(\eta\,;\, K \lhd_m v) : C_r}$$

We now continue with the computational rules for our abstract machine. The full set of rules can be found in Figure 5. We factor out passing a value to a match, $\eta\,;\, v \blacktriangleright_m M = \eta'\,;\, e'$ that produces a (possibly extended) environment η' and expression e'. In all cases below, we presuppose the variable names are chosen so the extended environment has unique bindings for each variable. For an extension with mutual recursion, see our extended version (link in preamble).

We obtain the following expected theorems of preservation and progress.

▶ **Theorem 15** (Preservation). *theorempreservation If $S : A$ and $S \longrightarrow S'$ then $S' : A$.*

Proof. By cases on $S \longrightarrow S'$, applying inversion to the typing of S and assembling a typing derivation of S' from the resulting information.

The trickiest case involves dereferencing a variable $x \mapsto v$ admitting contraction. It is sound because every variable y occurring in v must also admit contraction by monotonicity and, furthermore, such variables still have an occurrence in the value v that is being returned. Therefore in the typing of the environment we can now type $[x \mapsto v]$ with $[x : A_m]$. ◀

A machine state is *final* if it has the form $\eta\,;\, \epsilon \blacktriangleleft_m v$, that is, if a value is returned to the empty continuation in some global environment η. In order to prove progress, we need to characterize values of a given type using a *canonical forms* property. Note that we allow a context Δ to provide for the variables that may be embedded in a value of negative type ($\multimap$, $\&$, $\uparrow$), but that a variable by itself does not count as a value.

$$
\begin{array}{llllll}
\eta & ; & () & \blacktriangleright_m & (() \Rightarrow e') & = \eta ; e' \\
\eta & ; & (v_1, v_2) & \blacktriangleright_m & ((x_1, x_2) \Rightarrow e'(x_1, x_2)) & = \eta, x_1 \mapsto v_1, x_2 \mapsto v_2 ; e'(x_1, x_2) \\
\eta & ; & \ell(v) & \blacktriangleright_m & (\ell(x) \Rightarrow e'_\ell(x))_{\ell \in L} & = \eta, x \mapsto v ; e'_\ell(x) \\
\eta & ; & \mathbf{down}^n_m\, v & \blacktriangleright_m & (\mathbf{down}(x) \Rightarrow e'(x)) & = \eta, x \mapsto v ; e'(x)
\end{array}
$$

$$
\begin{array}{lcll}
\eta, x \mapsto v, \eta' \,;\, K \rhd_m x & \longrightarrow & \eta, \eta' \,;\, K \lhd_m v & (\mathsf{C} \notin \sigma(m)) \\
\eta, x \mapsto v, \eta' \,;\, K \rhd_m x & \longrightarrow & \eta, [x \mapsto v], \eta' \,;\, K \lhd_m v & (\mathsf{C} \in \sigma(m))
\end{array}
$$

$$
\begin{array}{lcll}
\eta \,;\, K \rhd_r \mathbf{match}_m\, e\, M_r & \longrightarrow & \eta \,;\, K \cdot (\mathbf{match}_m\, \_\, M_r) \rhd_m e & (\otimes, \mathbf{1}, \oplus, \downarrow) \\
\eta \,;\, K \cdot (\mathbf{match}_m\, \_\, M_r) \lhd_m v & \longrightarrow & \eta' \,;\, K \rhd_r e' \quad \text{where } \eta \,;\, v \blacktriangleright_m M_r = \eta' \,;\, e'
\end{array}
$$

$$
\begin{array}{lcll}
\eta \,;\, K \rhd_m \lambda x.\, e(x) & \longrightarrow & \eta \,;\, K \lhd_m \lambda x.\, e(x) & (\multimap) \\
\eta \,;\, K \rhd_m (e_1\, e_2) & \longrightarrow & \eta \,;\, K \cdot (\_\, e_2) \rhd_m e_1 & \\
\eta \,;\, K \cdot (\_\, e_2) \lhd_m v_1 & \longrightarrow & \eta \,;\, K \cdot (v_1\, \_) \rhd_m e_2 & \\
\eta \,;\, K \cdot (\lambda x.\, e(x), \_) \lhd_m v_2 & \longrightarrow & \eta, x \mapsto v \,;\, K \rhd_m e(x) &
\end{array}
$$

$$
\begin{array}{lcll}
\eta \,;\, K \rhd_m \{\ell \Rightarrow e_\ell\}_{\ell \in L} & \longrightarrow & \eta \,;\, K \lhd_m \{\ell \Rightarrow e_\ell\}_{\ell \in L} & (\&) \\
\eta \,;\, K \rhd_m e.\ell & \longrightarrow & \eta \,;\, K \cdot (\_.\ell) \rhd_m e & \\
\eta \,;\, K \cdot (\_.\ell) \lhd_m \{\ell \Rightarrow e_\ell\}_{\ell \in L} & \longrightarrow & \eta \,;\, K \rhd_m e_\ell & (\ell \in L)
\end{array}
$$

$$
\begin{array}{lcll}
\eta \,;\, K \rhd_m \mathbf{susp}^m_k\, e & \longrightarrow & \eta \,;\, K \lhd_m \mathbf{susp}^m_k\, e & (\uparrow) \\
\eta \,;\, K \rhd_k \mathbf{force}^m_k\, e & \longrightarrow & \eta \,;\, K \cdot (\mathbf{force}^m_k\, \_) \rhd_m e & \\
\eta \,;\, K \cdot (\mathbf{force}^m_k\, \_) \lhd_m \mathbf{susp}^m_k e & \longrightarrow & \eta \,;\, K \rhd_k e &
\end{array}
$$

$$
\begin{array}{lcll}
\eta \,;\, K \rhd_m (e_1, e_2) & \longrightarrow & \eta \,;\, K \cdot (\_, e_2) \rhd_m e_1 & (\otimes) \\
\eta \,;\, K \cdot (\_, e_2) \lhd_m v_1 & \longrightarrow & \eta \,;\, K \cdot (v_1, \_) \rhd_m e_2 & \\
\eta \,;\, K \cdot (v_1, \_) \lhd_m v_2 & \longrightarrow & \eta \,;\, K \lhd_m (v_1, v_2) &
\end{array}
$$

$$
\begin{array}{lcll}
\eta \,;\, K \rhd_m () & \longrightarrow & \eta \,;\, K \lhd_m () & (\mathbf{1})
\end{array}
$$

$$
\begin{array}{lcll}
\eta \,;\, K \rhd_m \ell(e) & \longrightarrow & \eta \,;\, K \cdot \ell(\_) \rhd_m e & (\oplus) \\
\eta \,;\, K \cdot \ell(\_) \lhd_m v & \longrightarrow & \eta \,;\, K \lhd_m \ell(v) &
\end{array}
$$

$$
\begin{array}{lcll}
\eta \,;\, K \rhd_m \mathbf{down}^n_m\, e & \longrightarrow & \eta \,;\, K \cdot \mathbf{down}^n_m\, \_ \rhd_n e & (\downarrow) \\
\eta \,;\, K \cdot (\mathbf{down}^n_m\, \_) \lhd_n v & \longrightarrow & \eta \,;\, K \lhd_m \mathbf{down}^n_m\, v &
\end{array}
$$

Figure 5 Computation Rules.

▶ **Lemma 16** (Canonical Forms). *If $\Delta \vdash v : A_m$ then one of the following applies:*

(i) *if $A_m = B_m \multimap C_m$ then $v = \lambda x.\, e(x)$ for some e*
(ii) *if $A_m = \&\{\ell : A^\ell_m\}_{\ell \in L}$ then $v = \{\ell \Rightarrow e_\ell\}_{\ell \in L}$ for some set e_ℓ*
(iii) *if $A_m = \uparrow^m_k B_k$ then $v = \mathbf{susp}^m_k\, e$*
(iv) *if $A_m = B_m \otimes C_m$ then $v = (v_1, v_2)$ for values v_1 and v_2*
(v) *if $A_m = \mathbf{1}$ then $v = ()$*
(vi) *if $A_m = \oplus\{\ell : B^\ell_m\}_{\ell \in L}$ then $v = \ell(v')$ for some $\ell \in L$ and value v'*
(vii) *if $A_m = \downarrow^n_m A_n$ then $v = \mathbf{down}^n_m v'$ for some value v'*

Proof. As usual, by inversion on typing and the possible forms of values, remembering that variables do not count as values. ◀

▶ **Theorem 17** (Progress). *theoremprogress If $S : C_r$ then either S is final or $S \mapsto S'$ for some S'*

Proof. By cases on the typing derivation for the configuration and inversion on the typing of the embedded frames, values, and expressions. We apply the canonical forms theorem when we need the shape of a value. ◀

Purely positive types play an important role because we view values of these types as *directly observable*, while values of negative types can only be observed indirectly through their elimination forms.

$$\text{Purely positive types} \quad A^+, B^+ \quad ::= \quad A^+ \otimes B^+ \mid \mathbf{1} \mid \oplus\{\ell : A^+_\ell\}_{\ell \in L} \mid \downarrow A^+$$

Values of purely positive types are closed, even if values of negative types may not be.

▶ **Lemma 18** (Positive Values). *If $\Delta \vdash v : A^+_r$ then $\cdot \vdash v : A^+_r$ and all declarations in Δ admit weakening (either due to their mode or because they are provisional).*

Proof. By induction on the structure of the typing derivation, recalling that variables are not values. ◀

We call a variable $x : A_m$ *linear* if $\sigma(m) = \{\,\}$, that is, the mode m admits neither weakening or contraction. We extend this term to types, bindings in the environment, etc. in the obvious way.

▶ **Theorem 19** (Freedom from Garbage). *theoremfreedom If $\cdot \vdash e : A^+_r$ and $\cdot\,;\, \epsilon \rhd_r e \longrightarrow^* \eta\,;\, \epsilon \blacktriangleleft_r v$, then η does not contain a binding $x \mapsto v$ with $\sigma(m) = \{\,\}$ where m is the mode of x.*

Proof. Because A^+_r is purely positive, we know by Lemma 18 that v is closed.

When the continuation K is empty, the typing rule for valid states implies that $\eta : \Delta$ and $\Delta \vdash v : A^+_r$ for some Δ. Since v is closed, Δ cannot contain any linear variables.

Then we prove by induction on the typing of η that none of variables in η can be linear. In the inductive case

$$\frac{\eta' : (\Delta\,;\,\Delta') \quad \Delta' \geq m \quad \Delta' \vdash v : A_m}{(\eta', x \mapsto v) : (\Delta, x : A_m)}$$

we know that m must admit weakening or contraction or both. Since $\Delta' \geq m$, by monotonicity, Δ' must also admit weakening or contraction and we can apply the induction hypothesis to $\eta' : (\Delta\,;\,\Delta')$. ◀

We call a variable x_m, an expression $e : A_m$, or a binding $x \mapsto v$ *strict* if $\sigma(m) \subseteq \{\mathsf{C}\}$, that is, m does not admit weakening.

▶ **Theorem 20** (Strictness). *theoremstrictness If $\cdot \vdash e : A^+_r$ and $\cdot\,;\, \epsilon \rhd_r e \longrightarrow^* \eta\,;\, \epsilon \blacktriangleleft_r v$, then every* strict *binding in η is of the form $[x \mapsto v]$.*

Proof. Because A^+_r is purely positive, we know by Lemma 18 that v is closed.

When the continuation K is empty, the typing rule for valid states implies $\eta : \Delta$ and $\Delta \vdash v : A^+_r$ for some Δ. Since v is closed, Δ contains strict variables only in the form $[x : A_m]$.

We prove by induction on the typing of η all strict variables in η have the form $[x \mapsto w]$. There are two inductive cases.

$$\frac{\eta' : (\Delta\,;\,\Delta') \quad \Delta' \geq m \quad \Delta' \vdash w : A_m}{(\eta', x \mapsto w) : (\Delta, x : A_m)}$$

Since m is not strict, it must admit weakening. Since $\Delta' \geq m$, every variable in Δ' must also admit weakening by monotonicity, so we can apply the induction hypothesis to $\Delta\,;\Delta'$.

$$\frac{\eta' : (\Delta\,;\Delta'_{\mathsf{W}}) \quad \Delta'_{\mathsf{W}} \geq m \quad \Delta'_{\mathsf{W}} \vdash w : A_m}{(\eta', [x \mapsto v]) : (\Delta, [x : A_m])}$$

Any declaration in Δ'_{W} either directly admits weakening or is of the form $[y : A_k]$ for a strict k so we can apply the induction hypothesis to $\eta' : (\Delta\,;\Delta'_{\mathsf{W}})$. ◀

In this context of call-by-value, this property expresses that every strict variable will be read at least once, since a binding $[x \mapsto v]$ arises only from reading the value of x. In call-by-need it means that the value is indeed needed.

▶ **Theorem 21** (Dead Code). *theoremdeadcode If* $\cdot \vdash e : A^+_r$ *and* $\cdot\,;\epsilon \rhd_r e \longrightarrow^* \eta\,;\epsilon \blacktriangleleft_r v$ *then every state during the computation either evaluates* $\rhd_m$ *or returns* $\blacktriangleleft_m$ *for* $m \geq r$.

Proof. Most rule do not change the subject's mode. Several rules potentially raise the mode, name evaluating a **match**, a **force**, or a **down**. For each of these there is a corresponding rule lowering the mode back to its original, namely return a value to a **match**, to a **force**, or to a **down**.

We say the mode of a frame f is the mode of the following state after a value is returned to f. We prove by induction over the computation that in all states, all continuation frames and subjects have modes $m \geq r$. ◀

▶ **Theorem 22** (Erasure). *corollaryerasure Assume* $\cdot \vdash e : A^+_r$ *and* $\cdot\,;\epsilon \rhd_r e \longrightarrow^* \eta\,;\epsilon \blacktriangleleft_r v$. *Let* Ω *be a new term of every type and no transition rule.*

If we obtain e' *by replacing all subterms of type* B_k *for* $k \not\geq r$ *with* Ω, *then evaluation* e' *still terminates in a final state. This final state differs from* v *in that subterms of mode* $k \not\geq r$ *are also replaced by* Ω.

Proof. The computation of e' parallels that of e. It would only get stuck for a state $\eta'\,;K' \rhd_k \Omega$, but that is impossible by the preceding dead code theorem since $k \not\geq r$. ◀

6 Algorithmic Type Checking

The bidirectional type system of Section 3 is not yet algorithmic, among other things because splitting a given context into $\Delta = (\Delta_1\,;\Delta_2)$ is nondeterministic. One standard solution is to track which hypotheses are used in one premise (which ends up Δ_1), subtract them from the available ones, and pass the remainder into the second premise (which ends up Δ_2 together with an overall remainder) [14]. This originated in proof search, but here when we actually have a proof terms available to check, other options are available. *Additive* resource management computes the *used* hypotheses (rather than the unused ones) and merges ("adds") them [37, 5], which is conceptually slightly simpler and also has been shown to be more efficient [28].

The main complication in the additive approach are internal and external choice, more specifically, the $\&R$ and $\oplus L$ rules when the choice is empty. For example, while checking $\Delta \vdash \{\,\} \Longleftarrow \&\{\,\}$ any subset of Δ could be used. We reuse the idea from the dynamics to have provisional hypotheses $[x : A_m]$. In the additive approach, the context merge for provisional hypotheses then no longer requires contraction since such variables do not occur (but could be considered as used). There are a plethora of different judgments, but it is not

clear how to simplify them. In defining the additive approach, the main two judgments are $\Gamma \vdash s \Longrightarrow A_m \mathbin{/} \Xi$ and $\Gamma \vdash e \Longleftarrow A_m \mathbin{/} \Xi$ which we summarize as $\Gamma \vdash e \Longleftrightarrow A_m \mathbin{/} \Xi$ where Γ is a plain (that is, free of provisional hypotheses) context containing *all* variables that might occur in e (regardless of mode or structural properties) and Ξ is a context that may contain provisional hypotheses. We maintain the mode invariant $\Xi \geq m$ (even if it may be the case that $\Gamma \not\geq m$). The rules can be found in Figure 6. We show some of the crucial properties to understand the rules, defining some of these operations later with these properties in mind.

Because we keep the contexts Δ free of provisional hypotheses, we define the relation $\Xi \sqsupseteq \Delta$ which may remove or keep provisional hypotheses.

$$
\begin{array}{rcllllll}
(\Xi, x : A_m) & \sqsupseteq & (\Delta, x : A) & \text{if} & \Xi & \sqsupseteq & \Delta \\
(\Xi, [x : A_m]) & \sqsupseteq & (\Delta, x : A) & \text{if} & \Xi & \sqsupseteq & \Delta \\
(\Xi, [x : A_m]) & \sqsupseteq & \Delta & \text{if} & \Xi & \sqsupseteq & \Delta \\
(\cdot) & \sqsupseteq & (\cdot) &
\end{array}
$$

With this relation, we can state the *soundness* of algorithmic typing.

▶ **Theorem 23** (Soundness of Algorithmic Typing)**.**
If $\Gamma \vdash e \Longleftrightarrow A_m \mathbin{/} \Xi$ *and* $\Xi \sqsupseteq \Delta$ *then* $\Delta \vdash e \Longleftrightarrow A_m$*.*

Proof. By rule induction on the algorithmic typing derivation and inversion of the $\Xi \sqsupseteq \Delta$ judgment. ◀

For completeness we need a different relation $\Delta \geq \Xi$ which means that Ξ contains a legal subset of the hypotheses in Δ. This means hypotheses in Δ might be in Ξ (possibly provisional) or not, but then only if they can be weakened.

$$
\begin{array}{rcllllll}
(\Delta, x : A_m) & \geq & (\Xi, x : A_m) & \text{if} & \Delta & \geq & \Xi & \\
(\Delta, x : A_m) & \geq & (\Xi, [x : A_m]) & \text{if} & \Delta & \geq & \Xi & \\
(\Delta, x : A_m) & \geq & \Xi & \text{if} & \Delta & \geq & \Xi & \text{provided } \mathsf{W} \in \sigma(m) \\
(\cdot) & \geq & (\cdot) &
\end{array}
$$

With this relation we can state the *completeness* of algorithmic typing.

▶ **Theorem 24** (Completeness of Algorithmic Typing)**.**
If $\Delta \vdash e \Longleftrightarrow A_m$ *then* $\Delta \vdash e \Longleftrightarrow A_m \mathbin{/} \Xi$ *for some* Ξ *with* $\Delta \geq \Xi$

Proof. By rule induction on the given bidirectional typing. ◀

For the algorithm itself we need several operations. Some key properties of these operations that are needed in the soundness and completness proof can be found in the extended version of this paper (link in preamble).

The first, $\Xi \setminus x : A$ removes $x : A$ from Ξ if this is legal operation. Its prototypical use is in the $\multimap I$ rule. For the rule application to be correct the new variable $x : A_m$ must either have been used and therefore occur in Ξ, or the mode m must allow weakening.

$$
\begin{array}{rcllll}
(\Xi, x : A_m) & \setminus & x : A_m & = & \Xi & \\
(\Xi, [x : A_m]) & \setminus & x : A_m & = & \Xi & \\
(\Xi, y : B_k) & \setminus & x : A_m & = & (\Xi \setminus x : A_m), y : B_k & \text{provided } y \neq x \\
(\Xi, [y : B_k]) & \setminus & x : A_m & = & (\Xi \setminus x : A_m), [y : B_k] & \text{provided } y \neq x \\
(\cdot) & \setminus & x : A_m & = & (\cdot) & \text{provided } \mathsf{W} \in \sigma(m)
\end{array}
$$

We also need two forms of context restriction. The first $\Xi\|_m$ removes all hypotheses whose mode is *not* greater or equal to m to restore our invariant. It fails if Ξ contains a *used* hypothesis B_r with $r \not\geq m$. It is used only in the $\uparrow I$ rule to restore the invariant.

The second form of context restriction occurs in the case of an empty internal or external choice. All of the hypothesis that are allowed by the independence principle could be considered used, but they might also not. We write $[\Gamma|_m]$. It is used only in the nullary case for internal and external choice.

$$
\begin{array}{rcll|rcll}
(\Xi, x : A_k)\|_m & = & \Xi\|_m, x : A_k & (k \geq m) & [(\Gamma, x : A_k)|_m] & = & [\Gamma|_m], [x : A_k] & (k \geq m) \\
(\Xi, [x : A_k])\|_m & = & \Xi\|_m, [x : A_k] & (k \geq m) & & & & \\
(\Xi, [x : A_k])\|_m & = & \Xi\|_m & (k \not\geq m) & [(\Gamma, x : A_k)|_m] & = & [\Gamma|_m] & (k \not\geq m) \\
(\cdot)\|_m & = & & (\cdot) & [(\cdot)|_m] & = & (\cdot) &
\end{array}
$$

We come to the final operation $\Xi_1 \sqcup \Xi_2$ which is needed for $\&I$ and $\oplus E$. Variables used in one branch must also be used in all other branches, or be available for weakening, either because they are provisional or because their mode admits weakening. This idea is captured formally by the definition of $\sqcup$.

$$
\begin{array}{rcll}
(\Xi_1, x : A_m) & \sqcup & (\Xi_2, x : A_m) & = & (\Xi_1 \sqcup \Xi_2), x : A_m & \\
(\Xi_1, [x : A_m]) & \sqcup & (\Xi_2, x : A_m) & = & (\Xi_1 \sqcup \Xi_2), x : A_m & \\
(\Xi_1, x : A_m) & \sqcup & (\Xi_2, [x : A_m]) & = & (\Xi_1 \sqcup \Xi_2), x : A_m & \\
(\Xi_1, [x : A_m]) & \sqcup & (\Xi_2, [x : A_m]) & = & (\Xi_1 \sqcup \Xi_2), [x : A_m] & \\
(\Xi_1, x : A_m) & \sqcup & \Xi_2 & = & (\Xi_1 \sqcup \Xi_2), x : A_m & \text{for } x \notin \mathrm{dom}(\Xi_2),\ \mathsf{W} \in \sigma(m) \\
\Xi_1 & \sqcup & (\Xi_2, x : A_m) & = & (\Xi_1 \sqcup \Xi_2), x : A_m & \text{for } x \notin \mathrm{dom}(\Xi_1),\ \mathsf{W} \in \sigma(m) \\
(\Xi_1, [x : A_m]) & \sqcup & \Xi_2 & = & \Xi_1 \sqcup \Xi_2 & \text{for } x \notin \mathrm{dom}(\Xi_2) \\
\Xi_1 & \sqcup & (\Xi_2, [x : A_m]) & = & \Xi_1 \sqcup \Xi_2 & \text{for } x \notin \mathrm{dom}(\Xi_1) \\
(\cdot) & \sqcup & (\cdot) & = & (\cdot) &
\end{array}
$$

7 Conclusion

We have presented a natural deduction formulation of adjoint logic. By carefully constructing these rules and the translations to and from the sequent calculus, we automatically obtained the presence of long normal forms for the proofs in natural deduction. We then presented a computational interpretation in the form of a state machine with a global context which leads to proofs of some properties of programs that come directly from having a mode hierarchy. Lastly, we presented an algorithmic type checking system, that due to the empty sum and (positive) product constructors requires a somewhat complicated approach.

There have been recent proposals to extend the adjoint approach to combining logics to dependent types. Licata et al. [34, 35] permit dependent types and richer connections between the logics that are combined, but certain properties such as independence are no longer fundamental and have to be proved in each case where they apply. While they mostly stay within a sequent calculus, they also briefly introduce natural deduction. They further provide a categorical semantics. Hanukaev and Eades [25] also permit dependent types and use the graded/algebraic approach to defining their system. However, their approach to dependency appears incompatible with control of contraction, so their adjoint structure is not nearly as general as ours. They also omit empty internal choice (and external choice altogether), which created some of the trickiest issues in our system. Curien et al. [19] investigate call-by-push-value [33] and provide a semantic foundation for the adjunction properties that is flexible enough to accommodate effects. It also incorporates Benton's mixed

$$\frac{\Gamma \vdash s \Longrightarrow A_m \;/\; \Xi}{\Gamma \vdash s \Longleftarrow A_m \;/\; \Xi}\;\Rightarrow/\Leftarrow \qquad \frac{\Gamma \vdash e \Longleftarrow A_m \;/\; \Xi}{\Gamma \vdash (e : A_m) \Longrightarrow A_m \;/\; \Xi}\;\Leftarrow/\Rightarrow \qquad \frac{x : A_m \in \Gamma}{\Gamma \vdash x \Longrightarrow A_m \;/\; (x : A_m)}\;\mathsf{hyp}$$

$$\frac{\Gamma, x : A_m \vdash e \Longleftarrow B_m \;/\; \Xi}{\Gamma \vdash \lambda x.\, e \Longleftarrow A_m \multimap B_m \;/\; (\Xi \setminus x : A_m)}\;\multimap I \qquad \frac{\Gamma \vdash s \Longrightarrow A_m \multimap B_m \;/\; \Xi \quad \Gamma \vdash e \Longleftarrow A_m \;/\; \Xi'}{\Gamma \vdash s\, e \Longrightarrow B_m \;/\; \Xi \,;\, \Xi'}\;\multimap E$$

$$\frac{}{\Gamma \vdash \{\,\} \Longleftarrow \&_m\{\,\} \;/\; [\Gamma|_m]}\;\&I_0 \qquad \frac{\Gamma \vdash e_\ell \Longleftarrow A^\ell_m \;/\; \Xi_\ell \quad (\forall \ell \in L \neq \emptyset)}{\Gamma \vdash \{\ell \Rightarrow e_\ell\}_{\ell \in L} \Longleftarrow \&\{\ell : A^\ell_m\}_{\ell \in L} \;/\; \sqcup_{\ell \in L} \Xi_\ell}\;\&I$$

$$\frac{\Gamma \vdash e \Longrightarrow \&\{\ell : A^\ell_m\}_{\ell \in L} \;/\; \Xi \quad (\ell \in L)}{\Gamma \vdash e.\ell \Longrightarrow A^\ell_m \;/\; \Xi}\;\&E$$

$$\frac{\Gamma \vdash e \Longleftarrow A_k \;/\; \Xi}{\Gamma \vdash \mathbf{susp}\; e \Longleftarrow \uparrow^m_k A_k \;/\; \Xi\|_m}\;\uparrow I \qquad \frac{\Gamma \vdash s \Longrightarrow \uparrow^m_k A_k \;/\; \Xi}{\Gamma \vdash \mathbf{force}\; s \Longrightarrow A_k \;/\; \Xi}\;\uparrow E$$

$$\frac{\Gamma \vdash e_1 \Longleftarrow A_m \;/\; \Xi \quad \Gamma \vdash e_2 \Longleftarrow B_m \;/\; \Xi'}{\Gamma \vdash (e_1, e_2) \Longleftarrow A_m \otimes B_m \;/\; \Xi \,;\, \Xi'}\;\otimes I$$

$$\frac{\Gamma \vdash s \Longrightarrow A_m \otimes B_m \;/\; \Xi \quad m \geq r \quad \Gamma, x_1 : A_m, x_2 : B_m \vdash e' \Longleftarrow C_r \;/\; \Xi'}{\Gamma \vdash \mathbf{match}\; s\; ((x_1, x_2) \Rightarrow e') \Longleftarrow C_r \;/\; \Xi \,;\, (\Xi' \setminus x_1 : A_m \setminus x_2 : B_m)}\;\otimes E$$

$$\frac{}{\Gamma \vdash (\,) \Longleftarrow \mathbf{1}_m \;/\; (\cdot)}\;\mathbf{1}I \qquad \frac{\Gamma \vdash s \Longrightarrow \mathbf{1}_m \;/\; \Xi \quad m \geq r \quad \Gamma \vdash e' \Longleftarrow C_r \;/\; \Xi'}{\Gamma \vdash \mathbf{match}\; s\; ((\,) \Rightarrow e') \Longleftarrow C_r \;/\; \Xi \,;\, \Xi'}\;\mathbf{1}E$$

$$\frac{\Gamma \vdash e \Longleftarrow A^\ell_m \;/\; \Xi \quad (\ell \in L)}{\Gamma \vdash \ell(e) \Longleftarrow \oplus\{\ell : A^\ell_m\}_{\ell \in L} \;/\; \Xi}\;\oplus I \qquad \frac{\Gamma \vdash s \Longrightarrow \oplus_m\{\,\} \;/\; \Xi \quad m \geq r}{\Gamma \vdash \mathbf{match}\; s\; (\,) \Longleftarrow C_r \;/\; \Xi \;;\; [\Gamma|_r]}\;\oplus E_0$$

$$\frac{\Gamma \vdash s \Longrightarrow \oplus\{\ell : A^\ell_m\}_{\ell \in L} \;/\; \Xi \quad m \geq r \quad \Gamma, x : A^\ell_m \vdash e_\ell \Longleftarrow C_r \;/\; \Xi'_\ell \quad (\forall \ell \in L \neq \emptyset)}{\Gamma \vdash \mathbf{match}\; s\; (\ell(x) \Rightarrow e_\ell)_{\ell \in L} \Longleftarrow C_r \;/\; \Xi \,;\, \sqcup_{\ell \in L} (\Xi'_\ell \setminus x : A^\ell_m)}\;\oplus E$$

$$\frac{\Gamma \vdash e \Longleftarrow A_n \;/\; \Xi}{\Gamma \vdash \mathbf{down}\; e \Longleftarrow \downarrow^n_m A_n \;/\; \Xi}\;\downarrow I \qquad \frac{\Gamma \vdash s \Longrightarrow \downarrow^n_m A_n \;/\; \Xi \quad m \geq r \quad \Gamma, x : A_n \vdash e' \Longleftarrow C_r \;/\; \Xi'}{\Gamma \vdash \mathbf{match}\; s\; (\mathbf{down}\; x \Rightarrow e') \Longleftarrow C_r \;/\; \Xi \,;\, (\Xi' \setminus x : A_n)}\;\downarrow E$$

Figure 6 Algorithmic Typing for Natural Deduction.

linear/nonlinear calculus [7] in the form of a sequent calculus but does not consider a general preorder of modes or more flexible structural properties. None of these propose an algorithm for type checking or an operational semantics that would exploit the substructural and mode properties to obtain "free theorems" about well-typed programs as in our dynamics.

We are pursuing several avenues building on the results of this paper. On the foundational side, we are looking for a direct algorithm to convert an arbitrary natural deduction into a verification. On the programming side, we are considering *mode polymorphism*: type-checking the same expression against multiple different modes to avoid code duplication. On the application side, we are considering staged computation, quotation, and metaprogramming, decomposing the usual type $\Box A$ or its contextual analogue along the lines of Example 8.

References

1 Andreas Abel, Nils Anders Danielsson, and Oskar Eriksson. A graded modal dependent type theory with a universe and erasure, formalized. *Proc. ACM Program. Lang.*, 7(ICFP'23):920–954, 2023.

2 Samson Abramsky. Computational interpretations of linear logic. *Theoretical Computer Science*, 111:3–57, 1993.

3 Beniamino Accattoli, Pablo Barenbaum, and Damiano Mazza. Distilling abstract machines. In J. Jeuring and M. Chakravarty, editors, *19th International Conference on Functional Programming (ICFP 2014)*, pages 363–376, Gothenburg, Sweden, September 2014. ACM. Long version available at `arXiv:1406.2370`.

4 Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):197–347, 1992.

5 Robert Atkey. Syntax and semantics of quantitative type theory. In Anuj Dawar and Erich Grädel, editors, *33rd Conference on Logic in Computer Science (LICS 2018)*, pages 56–65, Oxford, UK, July 2018. ACM.

6 Andrew Barber. Dual intuitionistic linear logic. Technical Report ECS-LFCS-96-347, Department of Computer Science, University of Edinburgh, September 1996.

7 P. N. Benton. A mixed linear and non-linear logic: Proofs, terms and models. In Leszek Pacholski and Jerzy Tiuryn, editors, *Selected Papers from the 8th International Workshop on Computer Science Logic (CSL'94)*, pages 121–135, Kazimierz, Poland, September 1994. Springer LNCS 933. An extended version appears as Technical Report UCAM-CL-TR-352, University of Cambridge.

8 P. N. Benton. A mixed linear and non-linear logic: Proofs, terms and models (preliminary report). Technical Report UCAM-CL-TR-352, University of Cambridge, October 1994. URL: `https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-352.html`.

9 P. N. Benton, G. M. Bierman, and V. C. V. de Paiva. Computational types from a logical perspective. *Journal of Functional Programming*, 8(2):177–193, March 1998.

10 P. N. Benton, Gavin M. Bierman, Martin Hyland, and Valeria de Paiva. A term calculus for intuitionistic linear logic. In *International Conference on Typed Lambda Calculi and Applications*, pages 75–90, Utrecht, The Netherlands, 1993. Springer LNCS 664.

11 P. N. Benton and Philip Wadler. Linear logic, monads, and the lambda calculus. In E. Clarke, editor, *Proceedings of the 11th Annual Symposium on Logic in Computer Science*, pages 420–431, New Brunswick, New Jersey, July 1996. IEEE Computer Society Press.

12 Luís Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In *Proceedings of the 21st International Conference on Concurrency Theory (CONCUR 2010)*, pages 222–236, Paris, France, August 2010. Springer LNCS 6269.

13 Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logic propositions as session types. *Mathematical Structures in Computer Science*, 26(3):367–423, 2016. Special Issue on Behavioural Types.

14 Iliano Cervesato, Joshua S. Hodas, and Frank Pfenning. Efficient resource management for linear logic proof search. *Theoretical Computer Science*, 232(1–2):133–163, February 2000. Special issue on Proof Search in Type-Theoretic Languages, D. Galmiche and D. Pym, editors.

15 Bor-Yuh Evan Chang, Kaustuv Chaudhuri, and Frank Pfenning. A judgmental analysis of linear logic. Technical Report CMU-CS-03-131R, Carnegie Mellon University, Department of Computer Science, December 2003.

16 Kaustuv Chaudhuri. Classical and intuitionistic subexponential logics are equally expressive. In *Computer Science Logic*, pages 185–199. Springer LNCS 6247, August 2010.

17 Pritam Choudhury, Harley Eades III, Richard A. Eisenberg, and Stephanie Weirich. A graded dependent type system with a usage-aware semantics. *Proc. ACM Program. Lang.*, 5(POPL'21):1–32, 2021.

18 Alonzo Church. *The Calculi of Lambda-Conversion*, volume 6 of *Annals of Mathematics Studies*. Princeton University Press, 1941.

19 Pierre-Louis Curien, Marcelo P. Fiore, and Guillaume Munch-Maccagnoni. A theory of effects and resources: Adjunction models and polarised calculi. In R. Bodík and R. Majumdar, editors, *Proceedings of the 43rd Symposium on Principles of Programming Languages (POPL 2016)*, pages 44–56, St. Petersburgh, Florida, USA, January 2016. ACM.

20 Vincent Danos, Jean Baptiste Joinet, and Harold Schellinx. The structure of exponentials: Uncovering the dynamics of linear logic proofs. In Georg Gottlob, Alexander Leitsch, and Daniele Mundici, editors, *Computational Logic and Proof Theory*, pages 159–171, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

21 Michael Dummett. *The Logical Basis of Metaphysics*. Harvard University Press, Cambridge, Massachusetts, 1991. The William James Lectures, 1976.

22 Gerhard Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935. English translation in M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131, North-Holland, 1969.

23 Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

24 Jean-Yves Girard and Yves Lafont. Linear logic and lazy computation. In H. Ehrig, R. Kowalski, G. Levi, and U. Montanari, editors, *Proceedings of the International Joint Conference on Theory and Practice of Software Development*, volume 2, pages 52–66, Pisa, Italy, March 1987. Springer-Verlag LNCS 250.

25 Peter Hanukaev and Harley Eades, III. Combining dependency, grades, and adjoint logic. In *8th Workshop on Type-Driven Development (TyDe 2023)*, pages 58–70, Seattle, Washington, 2023. ACM. `arXiv:2307.09563`.

26 Robert Harper. *Practical Foundations for Programming Languages*. Cambridge University Press, second edition, April 2016.

27 W. A. Howard. The formulae-as-types notion of construction. Unpublished note. An annotated version appeared in: *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, 479–490, Academic Press (1980), 1969.

28 Jack Hughes and Dominic Orchard. Resourceful program synthesis from graded linear types. In M. Fernández, editor, *30th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR 2020)*, pages 151–170, Bologna, Italy, September 2020. Springer LNCS 12561.

29 Max Kanovich, Stepan Kuznetsov, Vivek Nigam, and Andre Scedrov. Subexponentials in non-commutative linear logic. *ArXiv e-prints*, September 2017. `arXiv:1709.03607`.

30 Max Kanovich, Stepan Kuznetsov, Vivek Nigam, and Andre Scedrov. A logical framework with commutative and non-commutative subexponentials. In *International Joint Conference on Automated Reasoning (IJCAR 2018)*, pages 228–245. Springer LNAI 10900, 2018.

31 Stephen Cole Kleene. *Introduction to Metamathematics*. North-Holland, 1952.

32 John Launchbury. A natural semantics for lazy evaluation. In *20th Annual Symposium on Principles of Programming Languages (POPL 1993)*, pages 144–154, Charleston, South Carolina, January 1993. ACM.

33 Paul Blain Levy. Call-by-push-value: Decomposing call-by-value and call-by-name. *Higher-Order and Symbolic Computation*, 19(4):377–414, 2006.

34 Daniel R. Licata and Michael Shulman. Adjoint logic with a 2-category of modes. In *International Symposium on Logical Foundations of Computer Science (LFCS)*, pages 219–235. Springer LNCS 9537, January 2016.

35 Daniel R. Licata, Michael Shulman, and Mitchell Riley. A fibrational framework for substructural and modal logics. In Dale Miller, editor, *Proceedings of the 2nd International Conference on Formal Structures for Computation and Deduction (FSCD'17)*, pages 25:1–25:22, Oxford, UK, September 2017. LIPIcs.

36 Per Martin-Löf. On the meanings of the logical constants and the justifications of the logical laws. Notes for three lectures given in Siena, Italy. Published in *Nordic Journal of Philosophical Logic*, 1(1):11-60, 1996, April 1983. URL: `http://www.hf.uio.no/ifikk/forskning/publikasjoner/tidsskrifter/njpl/vol1no1/meaning.pdf`.

37 Conor McBride. I got plenty o' nuttin'. In Sam Lindley, Conor McBride, Phil Trinder, and Don Sannella, editors, *A List of Successes That can Change the World—Essays Dedicated to Philip Wadler on the Occasion of His 60th Birthday*, pages 207–233. Springer LNCS 9600, 2016.

38 Benjamin Moon, Harley Eades III, and Dominic Orchard. Graded modal dependent type theory. In *30th European Symposium on Programming (ESOP'21)*, volume 12648 of *Lecture Notes in Computer Science*, pages 462–490. Springer, 2021.

39 Alan Mycroft. The theory and practice of transforming call-by-need into call-by-value. In *4th International Symposium on Programming*, pages 269–281. Springer LNCS 83, 1980.

40 Vivek Nigam and Dale Miller. Algorithmic specifications in linear logic with subexponentials. In *Proceedings of the 11th International Conference on Principles and Practice of Declarative Programming (PPDP)*, pages 129–140, Coimbra, Portugal, September 2009. ACM.

41 Vivek Nigam, Elaine Pimental, and Giselle Reis. An extended framework for specifying and reasoning about proof systems. *Journal of Logic and Computation*, 26(2):539–576, 2016.

42 Frank Pfenning and Rowan Davies. A judgmental reconstruction of modal logic. *Mathematical Structures in Computer Science*, 11:511–540, 2001. Notes to an invited talk at the *Workshop on Intuitionistic Modal Logics and Applications* (IMLA'99), Trento, Italy, July 1999.

43 Frank Pfenning and Klaas Pruiksma. Relating message passing and shared memory, proof-theoretically. In S. Jongmans and A. Lopes, editors, *25th International Conference on Coordination Models and Languages (COORDINATION 2023)*, pages 3–27, Lisbon, Portugal, June 2023. Springer LNCS 13908. Notes to an invited talk.

44 Dag Prawitz. *Natural Deduction*. Almquist & Wiksell, Stockholm, 1965.

45 Klaas Pruiksma. *Adjoint Logic with Applications*. PhD thesis, Carnegie Mellon University, 2024. In preparation.

46 Klaas Pruiksma, William Chargin, Frank Pfenning, and Jason Reed. Adjoint logic. Unpublished manuscript, April 2018. URL: `http://www.cs.cmu.edu/~fp/papers/adjoint18b.pdf`.

47 Klaas Pruiksma and Frank Pfenning. A message-passing interpretation of adjoint logic. *Journal of Logical and Algebraic Methods in Programming*, 120(100637), 2021.

48 Klaas Pruiksma and Frank Pfenning. Back to futures. *Journal of Functional Programming*, 32:e6, 2022.

49 Jason Reed. A judgmental deconstruction of modal logic. Unpublished manuscript, May 2009. URL: `http://www.cs.cmu.edu/~jcreed/papers/jdml2.pdf`.

50 The Rust programming language. Available at `https://www.rust-lang.org/`. Accessed January 15, 2024.

51 Jesse A. Tov and Riccardo Pucella. Practical affine types. In T. Ball and M. Sagiv, editors, *Proceedings of the 38th Symposium on Principles of Programming Languages (POPL 2011)*, pages 447–458. ACM Press, January 2011.

52 Anne S. Troelstra. Natural deduction for intuitionistic linear logic. *Annals of Pure and Applied Logic*, 73(1):79–108, 1995.

53 Philip Wadler. Linear types can change the world. In *IFIP TC*, volume 2, pages 347–359, 1990.

54 Philip Wadler. Propositions as sessions. In *Proceedings of the 17th International Conference on Functional Programming (ICFP 2012)*, pages 273–286, Copenhagen, Denmark, September 2012. ACM Press.

55 James Wood and Robert Atkey. A framework for substructural type systems. In Ilya Sergey, editor, *31st European Symposium on Programming (ESOP 2022)*, pages 376–402, Munich, Germany, April 2022. Springer LNCS 13240.

On the Complexity of the Small Term Reachability Problem for Terminating Term Rewriting Systems

Franz Baader
Theoretical Computer Science, TU Dresden, Germany
SCADS.AI Dresden/Leipzig, Germany

Jürgen Giesl
RWTH Aachen University, Aachen, Germany

Abstract

Motivated by an application where we try to make proofs for Description Logic inferences smaller by rewriting, we consider the following decision problem, which we call the small term reachability problem: given a term rewriting system R, a term s, and a natural number n, decide whether there is a term t of size $\leq n$ reachable from s using the rules of R. We investigate the complexity of this problem depending on how termination of R can be established. We show that the problem is NP-complete for length-reducing term rewriting systems. Its complexity increases to N2ExpTime-complete (NExpTime-complete) if termination is proved using a (linear) polynomial order and to PSpace-complete for systems whose termination can be shown using a restricted class of Knuth-Bendix orders. Confluence reduces the complexity to P for the length-reducing case, but has no effect on the worst-case complexity in the other two cases.

2012 ACM Subject Classification Theory of computation → Equational logic and rewriting; Theory of computation → Automated reasoning; Theory of computation → Complexity theory and logic

Keywords and phrases Rewriting, Termination, Confluence, Creating small terms, Derivational complexity, Description Logics, Proof rewriting

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.16

Funding *Franz Baader*: Partially supported by DFG, Grant 389792660, within TRR 248 "Center for Perspicuous Computing", and by the German Federal Ministry of Education and Research (BMBF, SCADS22B) and the Saxon State Ministry for Science, Culture and Tourism (SMWK) by funding the competence center for Big Data and AI "ScaDS.AI Dresden/Leipzig".
Jürgen Giesl: Partially supported by DFG, Grant 235950644 (Project GI 274/6-2).

1 Introduction

Term rewriting [7, 28] is a well-investigated formalism, which can be used both for computation and deduction. A term rewriting system R consists of rules, which describe how a term s can be transformed into a new term t, in which case one writes $s \to_R t$. In the computation setting, where term rewriting is akin to functional programming [12], a given term (the input) is iteratively rewritten into a normal form (the output), which is a term that cannot be further rewritten. Termination of R prevents infinite rewrite chains, and thus ensures that a normal form can always be reached, whereas confluence guarantees that the output is unique, despite the nondeterminism inherent to the rewriting process (which rule to apply when and where). In the deduction setting, which is, e.g., relevant for first-order theorem proving with equality [25], one is interested in whether a term s can be rewritten into a term t by iteratively applying the rules of R in both directions. If R is confluent and terminating, this problem can be solved by computing normal forms of s and t, and then checking whether they are equal. In the present paper, we want to employ rewriting for a different purpose: given a term s, we are interested in finding a term t of minimal size that can be reached from s by rewriting (written $s \overset{*}{\to}_R t$), but this term need not be in normal form. To assess the

$$\mathsf{R1}\ \frac{A \sqsubseteq B \quad B \sqsubseteq C}{A \sqsubseteq C} \qquad \mathsf{R2}\ \frac{A \sqsubseteq B}{\exists r.A \sqsubseteq \exists r.B} \qquad \mathsf{R3}\ \frac{A \sqsubseteq \exists r.A_1 \quad A_1 \sqsubseteq B_1 \quad \exists r.B_1 \sqsubseteq B}{A \sqsubseteq B}$$

Figure 1 Three proof rules for $\mathcal{EL}$.

complexity of this computation problem, we investigate the corresponding decision problem: given a term rewriting system R, a term s, and a natural number n, decide whether there is a term t of size $\leq n$ such that $s \xrightarrow{*}_R t$. We call this the *small term reachability problem*.

Our interest in this problem stems from the work on finding small proofs [3, 4] for Description Logic (DL) inferences [6], which are then visualized in an interactive explanation tool [2]. For the DL $\mathcal{EL}$ [5], we employ the highly-efficient reasoner ELK [20] to compute proofs. However, the proof calculus employed by ELK is rather fine-grained, and thus produces relatively large proofs. Our idea was thus to generate smaller proofs by rewriting several proof steps into a single step. As a (simplified) example, consider the three proof rules in Figure 1. It is easy to see that one needs one application of R2 followed by two of R1 to produce the same consequence as a single application of R3. Thus, if one looks for patterns in a proof that use R1 and R2 in this way, and replaces them by the corresponding applications of R3, then one can reduce the size of a given proof. Given finitely many such proof rewriting rules and a proof, the question is then how to use the rules to rewrite the given proof into one of minimal size. Since tree-shaped proofs as well as DL concept descriptions can be represented as terms, this question can be seen as an instance of the small term reachability problem introduced above.

In this paper, we investigate the complexity of the small term reachability problem on the general level of term rewriting systems (TRSs). It turns out that this complexity depends on how termination of the given TRS can be shown. The paper contains the following main contributions:

1. Small term reachability for length-reducing TRSs

If the introduced rewrite rules are *length-reducing*, i.e., each rewrite step decreases the size of the term (proof), like the rule in our example, then termination of all rewrite sequences is guaranteed. In general, it may nevertheless be the case that one can generate two normal forms of different sizes. Confluence prevents this situation, i.e., then it is sufficient to generate only one rewrite sequence to produce a term (proof) of minimal size. In Section 4 we show that the small term reachability problem for length-reducing term rewriting systems is NP-complete in general, but becomes solvable in polynomial time if we restrict ourselves to confluent systems.

2. Small term reachability for TRSs whose termination is shown by polynomial orders

It also makes sense to consider sets of rules where not every rule is length-reducing, e.g., if one first needs to reshape a proof before a length-reducing rule can be applied, or if one translates between different proof calculi. In this extended setting, termination is no longer trivially given, and thus one first needs to show that the introduced set of rules is terminating, which can be achieved with the help of a reduction order [7, 28]. We show in this paper that the complexity of the small term reachability problem depends on which reduction order is

used for this purpose. More precisely, in Section 5 we consider term rewriting systems that can be proved terminating using a polynomial order [22], and show that in this case the small term reachability problem is N2ExpTime-complete, both in the general and the confluent case. If the definition of the polynomial order employs only linear polynomials, then the complexity of the problem is reduced to NExpTime, where again hardness already holds for confluent systems. Here, as usual, NExpTime (N2ExpTime) is the class of all decision problems solvable by a nondeterministic Turing machine in $O(2^{p(n)})$ ($O(2^{2^{p(n)}})$) steps, where n is the size of the problem and $p(n)$ is a polynomial in n.

3. Small term reachability for TRSs whose termination is shown by KBO

In Section 6, we investigate the impact that using a Knuth-Bendix order (KBO) [21] for the termination proof has on the complexity of the small term reachability problem. In the restricted setting without unary function symbols of weight zero, the problem is PSpace-complete, again both in the general and the confluent case. The complexity class PSpace consists of all decision problems solvable by a deterministic Turing machine in $O(p(n))$ space, where n is the size of the problem and $p(n)$ is a polynomial in n.

Our proofs of the results mentioned above strongly depend on work on the derivational complexity of term rewriting systems, which links the reduction order employed for the termination proof with the maximal length of reduction sequences as a function of the size of the start term (see e.g., [14, 15, 16, 23]). To obtain reasonable complexity classes, we restricted ourselves to reduction orders where the resulting bound on the derivational complexity is not "too high". In particular, we use the results of the seminal paper by Hofbauer and Lautemann [16], which show that termination proofs with a (linear) polynomial order yield a double-exponential (exponential) upper bound on the length of derivation sequences whereas termination proofs with a KBO without unary function symbols of weight zero yield an exponential such bound. We also make use of the term rewriting systems employed in the proofs showing that these bounds are tight. A connection between the derivational complexity of term rewriting systems and complexity classes has been established in [9] for polynomial orders and in [10] for Knuth-Bendix orders. While this work considers a different problem since it views term rewriting systems as devices for computing functions by generating a normal form, and uses them to characterize complexity classes, the constructions utilized in the proofs in [9, 10] are similar to the ones we use in our hardness proofs. A notable difference between the two problems is the impact that confluence has on the obtained complexity class: while in our setting confluence only reduces the complexity in the case of length-reducing systems, in [9] it also reduces the complexity (from the nondeterministic to the respective deterministic class) for the case of systems shown terminating with a (linear) polynomial order.

In the next section, we briefly recall basic notions from term rewriting, including the definitions of polynomial and Knuth-Bendix orders. In Section 3, we introduce the small term reachability problem and show that it is undecidable in general, but decidable for terminating systems. Sections 4, 5, and 6 respectively consider the length-reducing, polynomial order, and Knuth-Bendix order case. We conclude with a brief discussion of possible future work.

2 Preliminaries

We assume that the reader is familiar with basic notions and results regarding term rewriting. In this section, we briefly recall the relevant notions, but refer the reader to [7, 28] for details.

Given a finite set of *function symbols* with associated *arities* (called the *signature*) and a disjoint set of *variables*, terms are built in the usual way. Function symbols of arity 0 are also called *constant symbols*. For example, if x, y are variables, c is a constant symbol, and f a binary function symbol, then $c, f(x,c), f(f(x,c),c)$ are terms. The *size* $|t|$ of a term t is the number of occurrences of functions symbols and variables in t (e.g., $|f(f(x,c),c)| = 5$). If f is a function symbol or variable, then $|t|_f$ counts the number of occurrences of f in t (e.g., $|f(f(x,c),c)|_f = 2$). As usual, nested applications of unary function symbols are often written as words. For example, $g(g(h(h(g(x)))))$ is written as $gghhg(x)$ or $g^2h^2g(x)$.

A *rewrite rule* (or simply rule) is of the form $l \to r$ where l, r are terms such that l is not a variable and every variable occurring in r also occurs in l. In this paper, a term rewriting system (TRS) is a *finite* set of rewrite rules, and thus we do not mention finiteness explicitly when formulating our complexity results. A given TRS R induces the binary relation $\to_R$ on terms. Basically, we have $s \to_R t$ if there is a rule $l \to r$ in R such that s contains a substitution instance $\sigma(l)$ of l as subterm, and t is obtained from s by replacing this subterm with $\sigma(r)$. Recall that a *substitution* is a mapping from variables to terms, which is homomorphically extended to a mapping from terms to terms. For example, if R contains the rule $hh(x) \to g(x)$, then $f(hhh(c),c) \to_R f(gh(c),c)$ and $f(hhh(c),c) \to_R f(hg(c),c)$. The reflexive and transitive closure of $\to_R$ is denoted as $\xrightarrow{*}_R$, i.e., $s \xrightarrow{*}_R t$ holds if there are $n \geq 1$ terms $t_1, \ldots, t_n$ such that $s = t_1, t = t_n$, and $t_i \to_R t_{i+1}$ for $i = 1, \ldots, n-1$.

Two terms s_1, s_2 are *joinable* with R if there is a term t such that $s_i \xrightarrow{*}_R t$ holds for $i = 1, 2$. The relation $\to_R$ is *confluent* if $s \xrightarrow{*}_R s_i$ for $i = 1, 2$ implies that s_1 and s_2 are joinable with R. It is *terminating* if there is no infinite reduction chain $t_0 \to_R t_1 \to_R t_2 \to_R \ldots$. If $\to_R$ is confluent (terminating), then we also call R confluent (terminating). The term t is *irreducible* if there is no term t' such that $t \to_R t'$. If $s \xrightarrow{*}_R t$ and t is irreducible, then we call t a *normal form* of s. If R is confluent and terminating, then every term has a unique normal form. If R is terminating, then its confluence is decidable [21]. Termination can be proved using a *reduction order*, which is a well-founded order $\succ$ on terms such that $l \succ r$ for all $l \to r \in R$ implies $s \succ t$ for all terms s, t with $s \to_R t$. Since $\succ$ is well-founded, this then implies termination of R. If $l \succ r$ holds for all $l \to r \in R$, then we say that R can be *shown terminating* with the reduction order $\succ$. The following is a simple reduction order.

▶ **Example 1.** If we define $s \succ t$ if $|s| > |t|$ and $|s|_x \geq |t|_x$ for all variables x, then $\succ$ is a reduction order (see Exercise 5.5 in [7]). For example, $hh(x) \succ g(x)$, and thus the TRS $R = \{hh(x) \to g(x)\}$ is terminating. As illustrated in Example 5.2.2 in [7], the condition on variables is needed to obtain a reduction order.

This order can only show termination of *length-reducing* TRSs R, i.e., where $s \to_R t$ implies $|s| > |t|$. We now recapitulate the definitions of more powerful reduction orders [7, 28].

Polynomial orders

To define a polynomial order, one assigns to every n-ary function symbol f a polynomial P_f with coefficients in the natural numbers $\mathbb{N}$ and n indeterminates such that P_f depends on all these indeterminates. To ensure that this implies (strong) monotonicity of the polynomial order, we require that constant symbols c must be assigned a polynomial of degree 0 whose coefficient is > 0. Such an assignment also yields an assignment of polynomials P_t to terms t.

▶ **Example 2.** Assume that $+$ is binary, s, d, q are unary, and 0 is a constant. We assign the polynomial $P_+ = x + 2y + 1$ to $+$, $P_s = x + 2$ to s, $P_d = 3x + 1$ to d, $P_q = 3x^2 + 3x + 1$ to q, and $P_0 = 3$ to 0. For the terms $l = q(s(x))$ and $r = q(x) + s(d(x))$ we then obtain the associated polynomials $P_l = 3(x+2)^2 + 3(x+2) + 1 = 3x^2 + 15x + 19$ and $P_r = 3x^2 + 3x + 1 + 2(3x + 1 + 2) + 1 = 3x^2 + 9x + 8$.

The polynomial order induced by such an assignment is defined as follows: $t \succ t'$ if P_t evaluates to a larger natural number than $P_{t'}$ for every assignment of natural numbers > 0 to the indeterminates of P_t and $P_{t'}$. In our example, the evaluation of P_l is obviously always larger than the evaluation of P_r, and thus $l \succ r$. As shown, e.g., in Section 5.3 of [7], polynomial orders are reduction orders, and thus can be used to prove termination of TRSs.

Knuth-Bendix orders

To define a Knuth-Bendix order (KBO), one must assign a weight $w(f)$ to all function symbols and variables, and define a strict order $>$ on the function symbols (called *precedence*) such that the following is satisfied:

- All weights $w(f)$ are non-negative real numbers, and there is a weight $w_0 > 0$ such that $w(x) = w_0$ for all variables x and $w(c) \geq w_0$ for all constant symbols c.
- If there is a unary function symbol h with $w(h) = 0$, then h is the greatest element w.r.t. $>$, i.e., $h > f$ for all function symbols $f \neq h$. Such a unary function symbol h is then called a *special* symbol. Obviously, there can be at most one special symbol.

Since in this paper we only consider KBOs without special symbol, we restrict our definition of KBOs to this case. A given weight function w and strict order $>$ without special symbol induces the following KBO $\succ$: $s \succ t$ if $|s|_x \geq |t|_x$ for all variables x and

- $w(s) > w(t)$, where $w(u) := \sum_{f \text{ occurs in } u} w(f) \cdot |u|_f$ for all terms u, or
- $w(s) = w(t)$ and one of the following two conditions is satisfied:
  - $s = f(s_1, \ldots, s_m)$, $t = g(t_1, \ldots, t_n)$, and $f > g$.
  - $s = f(s_1, \ldots, s_m)$, $t = f(t_1, \ldots, t_m)$, and there is $i, 1 \leq i \leq m$, such that $s_1 = t_1, \ldots, s_{i-1} = t_{i-1}$, and $s_i \succ t_i$.

A proof of the fact that KBOs are reduction orders can, e.g., be found in Section 5.4.4 of [7].

▶ **Example 3.** Let $0, 1, 1'$ be unary function symbols and c a constant symbol, and consider the following TRS, which is similar to the one introduced in the proof of Lemma 7 in [10]:

$$R = \{1(c) \to 0(c), 0(c) \to 1'(c), 0(1'(x)) \to 1'(1(x)), 1(1'(x)) \to 0(1(x))\}.$$

Basically, this TRS realizes a binary down counter, and thus it is easy to see that, starting with the binary representation $10^n(c)$ of the number 2^n, the TRS R can make $\geq 2^n$ reduction steps to arrive at the term $0^{n+1}(c)$. For example, $100(c) \to_R 101'(c) \to_R 11'1(c) \to_R 011(c) \to_R 010(c) \to_R 011'(c) \to_R 001(c) \to_R 000(c)$. Termination of R can be shown using the following KBO: assign weight 1 to all function symbols and variables, and use the precedence order $1 > 0 > 1'$.

3 Problem definition and (un)decidability results

In this paper, we investigate the complexity of the following decision problem.

▶ **Definition 4.** *Given a TRS R, a term s, and a natural number n, the* small term reachability problem *asks whether there exists a term t such that $s \xrightarrow{*}_R t$ and $|t| \leq n$.*

The name "small term reachability problem" is motivated by the fact that we want to use the TRS R to turn a given term s into a term whose size is as small as possible. The introduced problem is the decision variant of this computation problem. A solution to the computation problem, which computes a term t of minimal size reachable with R from s, of course also solves the decision variant of the problem. Thus, complexity lower bounds for the decision problem transfer to the computation problem.

It is easy to see that this problem is in general undecidable, but decidable for terminating TRSs. For non-terminating systems, confluence is not sufficient to obtain decidability.

▶ **Proposition 5.** *The small term reachability problem is in general undecidable for confluent TRSs, but is decidable for systems that are terminating.*

Proof. Undecidability in the general case follows, e.g., from the fact that TRSs can simulate Turing machines (TMs) [17]. (We will also use Turing machines for the proofs of the hardness results in the remainder of the paper.) More precisely, the reduction introduced in Section 5.1.1 of [7] transforms a given TM $\mathcal{M}$ into a TRS $R_{\mathcal{M}}$ such that (among other things) the following holds: there is an infinite run of $\mathcal{M}$ on the empty input iff there is an infinite reduction sequence of $R_{\mathcal{M}}$ starting with the term s_0 that encodes the initial configuration of $\mathcal{M}$ for the empty input. In addition, if $\mathcal{M}$ is deterministic, then $R_{\mathcal{M}}$ is confluent. We can now add rules to $R_{\mathcal{M}}$ that apply to all terms encoding a halting configuration of $\mathcal{M}$, and trigger further rules that reduce such a term to one of size 1. Since the term s_0 has size larger than one and the rules of $R_{\mathcal{M}}$ never decrease the size of a term, this yields a reduction of the (undecidable) halting problem for deterministic TMs to the small term reachability problem for confluent TRSs.

Given a terminating TRS R and a term s, we can systematically generate all terms reachable from s by iteratively applying $\to_R$. Since R is finite, $\to_R$ is finitely branching. Together with termination, this means (by König's Lemma) that there are only finitely many terms reachable with R from s (see Lemma 2.2.4 in [7]). We can then check whether, among them, there is a term of size at most n. ◀

In the following, we study the *complexity* of the small term reachability problem for terminating TRSs, depending on how their termination can be shown.

4 Length-reducing term rewriting systems

In this section, we investigate the complexity of the small term reachability problem for length-reducing TRSs, i.e., TRSs where each rewrite step decreases the size of the term.

We start with showing an *NP upper bound*. Let R, s, n be an instance of the small term reachability problem, where R is assumed to be length-reducing. This assumption implies that the length k of any rewrite sequence $s \to_R s_1 \to_R s_2 \to_R \ldots \to_R s_k$ issuing from s is bounded by $|s|$. In addition, for each term s_i there are only polynomially many terms s' (in the size of s and R) such that $s_i \to_R s'$. Thus, the following yields an NP procedure for deciding the small term reachability problem:

- guess a rewrite sequence $s \to_R s_1 \to_R s_2 \to_R \ldots \to_R s_k$ of length $k \leq |s|$;
- check whether $|s_k| \leq n$ holds. If the answer is "yes" then accept, and reject otherwise.

▶ **Lemma 6.** *The small term reachability problem is in NP for length-reducing TRSs.*

If the length-reducing system R is confluent, then it is sufficient to generate an arbitrary *terminating* (i.e., maximal) rewrite sequence starting in s, i.e., a sequence $s \to_R s_1 \to_R s_2 \to_R \ldots \to_R s_k$ such that s_k is irreducible. Obviously, we have $k \leq |s|$, and thus such a sequence can be generated in polynomial time. We claim that there is a term t of size $\leq n$ reachable from s iff $|s_k| \leq n$. Otherwise, the smallest term t reachable from s is different from s_k. But then t and s_k are both reachable from s, and thus must be joinable due to the confluence of R. As s_k is irreducible, this implies $t \to_R^* s_k$ and thus, $|t| \geq |s_k|$, i.e., t is not smaller than s_k.

▶ **Proposition 7.** *For confluent length-reducing TRSs, the small term reachability problem can be decided in deterministic polynomial time.*

In general, however, the problem is NP-hard. We prove *NP-hardness* by showing that any polynomially time bounded nondeterministic Turing machine can be simulated by a length-reducing TRS. Thus, assume that $\mathcal{M}$ is such a TM and that its time-bound is given by the polynomial p. As in [7] we assume that in every step $\mathcal{M}$ either moves to the left or to the right, where the tape of the TM is infinite in both directions. In addition, we assume without loss of generality that $\mathcal{M}$ has exactly one accepting state $\widehat{q}$. We view the tape symbols of $\mathcal{M}$ as unary function symbols and the states of $\mathcal{M}$ as binary function symbols. We assume that q_0 is the initial state of $\mathcal{M}$ and that b is the blank symbol. Furthermore, let $\#$ be a constant symbol and f be a unary function symbol different from the tape symbols.

Given an input word $w = a_1 \dots a_\ell$ for $\mathcal{M}$, we construct the term

$$t(w) := b^{p(\ell)}(q_0(a_1 \dots a_\ell b^{p(\ell)-\ell}(\#), f^{p(\ell)}(\#))).$$

Intuitively, the starting b symbols together with the first argument of q_0 in $t(w)$ provide a tape that is large enough for a $p(\ell)$-time bounded TM to run on for the given input w of length ℓ. The first argument of a state symbol represents the part of the tape that starts at the position of the head. Thus, in $t(w)$, a_1 is the tape symbol at the position of the head and $a_2 \dots a_\ell$ are the symbols to the right of it. The second argument of a state symbol is a unary down counter from which one f is removed in every step that $\mathcal{M}$ makes. This is needed to ensure that the constructed TRS is length-reducing. This counter is large enough to allow $\mathcal{M}$ to make the maximal possible number of $p(\ell)$ steps.

Basically, we now express the transitions of $\mathcal{M}$ as usual by rewrite rules (as, e.g., done in Definition 5.1.3 of [7]), but with three differences:

- since the term $t(w)$ provides enough tape for a TM that can make at most $p(\ell)$ steps, the special cases that treat a situation where the end of the represented tape is reached and one has to add a blank are not needed;
- since we fix as start term $t(w)$ a configuration term (i.e., a term that encodes a configuration of the TM), the additional effort expended in [7] to deal with non-configuration terms (by using copies of symbols with arrows to the left or right) is not needed;
- we have the additional counter in the second argument, which removes one f in every step, and thus ensures that rule application is length-reducing.

The TRS $R^{\mathcal{M}}$ that simulates $\mathcal{M}$ has the following rewriting rules:

- For each transition (q, a, q', a', r) of $\mathcal{M}$ it has the rule $q(a(x), f(y)) \to a'(q'(x, y))$. Thus, the tape symbol a is replaced by a' and the head of the TM is now at the position to the right of it.
- For each transition (q, a, q', a', l) of $\mathcal{M}$ it has the rule $c(q(a(x), f(y))) \to q'(ca'(x), y)$ for every tape symbol c of $\mathcal{M}$. Thus, a is replaced by a' and the head of the TM is now at the position to the left of it.

Note that the blank symbol b is also considered as a tape symbol of $\mathcal{M}$.

In addition, we add rules to $R^{\mathcal{M}}$ that can be used to generate the term $\#$, which has size 1, whenever $\widehat{q}$ is reached:

- $a(\widehat{q}(x, y)) \to \widehat{q}(x, y)$ for every tape symbol a of $\mathcal{M}$,
- $\widehat{q}(x, y) \to \#$.

The following is now easy to see.

▶ **Lemma 8.** *The term $t(w)$ can be rewritten with $R^{\mathcal{M}}$ to a term of size 1 iff $\mathcal{M}$ accepts the word w.*

Proof. It is easy to see that $R^{\mathcal{M}}$ simulates $\mathcal{M}$ in the sense that there is a run of $\mathcal{M}$ on input $w = a_1 \dots a_\ell$ that reaches the accepting state $\widehat{q}$ iff there is a rewrite sequence of $R^{\mathcal{M}}$ starting with $t(w)$ that reaches a term of the form $u(\widehat{q}(t, t'))$, where u is a word over the tape symbols of $\mathcal{M}$ and t, t' are terms. Note that the assumption that $\mathcal{M}$ is $p(\ell)$-time bounded together with the construction of $t(w)$ ensures that there is enough tape space and the counter is large enough for the simulation of $\mathcal{M}$ to run through completely.

Thus, if $\mathcal{M}$ accepts $w = a_1 \dots a_\ell$, then we can rewrite $t(w)$ with $R^{\mathcal{M}}$ into a term of the form $u(\widehat{q}(t, t'))$, and this term can then be further rewritten into $\#$, which has size 1. If $\mathcal{M}$ does not accept $w = a_1 \dots a_\ell$, then the state $\widehat{q}$ cannot be reached by any run of $\mathcal{M}$ starting with this word. Thus, all terms reachable from $t(w)$ with the rules of $R^{\mathcal{M}}$ that simulate $\mathcal{M}$ are of the form $u(q(t, t'))$ for states q different from $\widehat{q}$. The rules of $R^{\mathcal{M}}$ of the second kind are thus not applicable, and the terms of the form $u(q(t, t'))$ clearly have size > 1. ◀

We are now ready to show the corresponding complexity lower bound.

▶ **Lemma 9.** *The small term reachability problem for length-reducing TRSs is NP-hard.*

Proof. We show that every problem Π in NP can be reduced in polynomial time to our problem. Let $\mathcal{M}$ be the nondeterministic Turing machine that is an NP decision procedure for Π, and let p be the polynomial that bounds the length of runs of $\mathcal{M}$. We can construct the length-reducing TRS $R^{\mathcal{M}}$ as described above. Given a word $w = a_1 \dots a_\ell$, we can compute the term $t(w)$ in polynomial time, and Lemma 8 implies that this yields a reduction function from Π to the small term reachability problem for the length-reducing TRS $R^{\mathcal{M}}$. ◀

Combining the obtained upper and lower bounds, we thus have determined the exact complexity of the problem under consideration.

▶ **Theorem 10.** *The small term reachability problem is NP-complete for length-reducing TRSs.*

To show that a given TRS R is length-reducing, one can, for example, use the reduction order of Example 1. This order also applies to the TRS $R^{\mathcal{M}}$ introduced above.

5 Term rewriting systems shown terminating with a polynomial order

An interesting question is whether similar results can be obtained for TRSs whose termination can be shown using a reduction order from a class of such orders that provides an upper bound on the length of reduction sequences. For example, it is known that a proof of termination using a polynomial order yields a double-exponential upper bound on the length of reduction sequences [16]. One possible conjecture could now be that, for TRSs whose termination can be shown using a polynomial order, the small term reachability problem is N2ExpTime-complete.

The *upper bound* is easy to show since, again, one just needs to guess a reduction sequence, but now of double-exponential length, and then check the size of the obtained term. This yields a nondeterministic double-exponential time procedure for solving the small term reachability problem for TRSs whose termination can be shown using a polynomial order.

▶ **Lemma 11.** *The small term reachability problem is in N2ExpTime for TRSs whose termination can be shown using a polynomial order.*

Regarding the *lower bound*, the idea is now to use basically the same approach as employed in Section 4, but generate a double-exponentially large tape and a double-exponentially large counter with the help of a TRS whose termination can be shown using a polynomial order.

For this, we want to re-use the original system introduced by Hofbauer and Lautemann showing that the double-exponential upper bound is tight (see Example 5.3.12 in [7]).

▶ **Example 12.** Let R_{HL} be the TRS consisting of the following rules:

$$\begin{array}{llll} x+0 \to x, & x+s(y) \to s(x+y), & d(0) \to 0, & d(s(x)) \to s(s(d(x))), \\ q(0) \to 0, & q(s(x)) \to q(x)+s(d(x)). & & \end{array}$$

The TRS R_{HL} intuitively defines the arithmetic functions addition ($+$), double (d), and square (q) on non-negative integers. Thus, it is easy to see that the term $t_n := q^n(s^2(0))$ can be reduced to $s^{2^{2^n}}(0)$. The polynomial order in Example 2 shows termination of R_{HL}.

Now, assume that $\mathcal{M}$ is a double-exponentially time bounded nondeterministic TM and that its time-bound is $2^{2^{p(\ell)}}$ for a polynomial p, where ℓ is the length of the input word. Given an input word $w = a_1 \dots a_\ell$ for $\mathcal{M}$, we construct the term

$$t(w) := q_1^{p(\ell)}(bb(q_0(a_1 \dots a_\ell q_2^{p(\ell)}(bb(\#)), q_3^{p(\ell)}(ff(\#))))).$$

The idea underlying this definition is that the term $q_1^{p(\ell)}(bb(q_0(\cdot))$ can be used to generate a tape segment before the read-write head of the TM (marked by the state q_0) with $2^{2^{p(\ell)}}$ blanks using the following modified version of R_{HL}:

$$\begin{array}{lll} R_1 := \{\, q_0(y_1,y_2) +_1 q_0(z_1,z_2) \to q_0(y_1,y_2), & b(x) +_1 q_0(z_1,z_2) \to b(x), & x +_1 b(y) \to b(x +_1 y), \\ \quad d_1(q_0(z_1,z_2)) \to q_0(z_1,z_2), & d_1(b(x)) \to b(b(d_1(x))), & \\ \quad q_1(q_0(z_1,z_2)) \to q_0(z_1,z_2), & q_1(b(x)) \to q_1(x) +_1 b(d_1(x))\}. & \end{array}$$

Here b plays the rôle of the successor function s in R_{HL}, terms of the form $q_0(\cdot)$ play the rôle of the zero 0 in R_{HL}, and $+_1$, d_1, and q_1 correspond to addition, double, and square. Instead of the rule $x +_1 q_0(z_1,z_2) \to x$ we considered two rules for the case where x is built with q_0 or with b, respectively. The reason will become clear later when we consider the restriction to confluent TRSs. Lemma 13 is an easy consequence of our observations regarding R_{HL}.

▶ **Lemma 13.** *For any two terms t_1, t_2, we can rewrite the term $q_1^{p(\ell)}(bb(q_0(t_1,t_2)))$ with R_1 into the term $b^{2^{2^{p(\ell)}}}(q_0(t_1,t_2))$.*

Next, we define a copy of R_{HL} that allows us to create a tape segment with $2^{2^{p(n)}}$ blanks to the right of the input word:

$$\begin{array}{lll} R_2 := \{\#+_2\#\to\#, & b(y)+_2\#\to b(y), & x+_2 b(y)\to b(x+_2 y), \\ \quad d_2(\#)\to\#, & d_2(b(x))\to b(b(d_2(x))), & \\ \quad q_2(\#)\to\# & q_2(b(x))\to q_2(x)+_2 b(d_2(x))\}. & \end{array}$$

▶ **Lemma 14.** *The term $q_2^{p(\ell)}(bb(\#))$ rewrites with R_2 to the term $b^{2^{2^{p(\ell)}}}(\#)$.*

The double-exponentially large counter can be generated by the following copy of R_{HL}:

$$\begin{array}{lll} R_3 := \{\#+_3\#\to\#, & f(y)+_3\#\to f(y), & x+_3 f(y)\to f(x+_3 y), \\ \quad d_3(\#)\to\#, & d_3(f(x))\to f(f(d_3(x))), & \\ \quad q_3(\#)\to\# & q_3(f(x))\to q_3(x)+_3 f(d_3(x))\}. & \end{array}$$

▶ **Lemma 15.** *The term $q_3^{p(\ell)}(ff(\#))$ rewrites with R_3 to the term $f^{2^{2^{p(\ell)}}}(\#)$.*

We now add to these three TRSs the system $R^{\mathcal{M}}$, which can simulate $\mathcal{M}$ and then make the term small in case the accepting state $\widehat{q}$ is reached. For the following lemma we assume, as before, that $\widehat{q}$ is the only accepting state. In addition, we assume without loss of generality that the initial state q_0 is not reachable, i.e., as soon as the machine has made a transition, it is in a state different from q_0 and cannot reach state q_0 again.

▶ **Lemma 16.** *The term $t(w)$ can be rewritten with $R^{\mathcal{M}} \cup R_1 \cup R_2 \cup R_3$ to a term of size 1 iff $\mathcal{M}$ accepts the word w.*

Proof. First, assume that $\mathcal{M}$ accepts the word w. Then there is a run of $\mathcal{M}$ on input w such that the accepting state $\widehat{q}$ is reached. We can simulate this run, starting with $t(w)$ by first using $R_1 \cup R_2 \cup R_3$ to generate the term

$$b^{2^{2^{p(\ell)}}}(q_0(a_1 \dots a_\ell b^{2^{2^{p(\ell)}}}(\#), f^{2^{2^{p(\ell)}}}(\#))).$$

Since the tape and counter generated this way are large enough, $R^{\mathcal{M}}$ can then simulate the accepting run of $\mathcal{M}$, and the last two rules of $R^{\mathcal{M}}$ can be used to generate the term $\#$, which has size 1.

For the other direction, we first note that a term of size 1 can only be reached from $t(w)$ using $R^{\mathcal{M}} \cup R_1 \cup R_2 \cup R_3$ if a term is reached that contains $\widehat{q}$. This function symbol can only be generated by performing transitions of $\mathcal{M}$, starting with the input w. In fact, while the simulation of $\mathcal{M}$ can start before the system $R_1 \cup R_2 \cup R_3$ has generated the tape and the counter in full size, rules of $R^{\mathcal{M}}$ can only be applied if the TM locally sees a legal tape configuration. This means that blanks generated by R_1 and R_2 can be used even if the application of these systems has not terminated yet. But if one of the auxiliary symbols employed by these systems is encountered, then no rule simulating a transition of $\mathcal{M}$ is applicable. These systems cannot generate tape symbols other than blanks, and these blanks are also available to $\mathcal{M}$ in its run. Thus, $R^{\mathcal{M}} \cup R_1 \cup R_2 \cup R_3$ can only generate a term containing $\widehat{q}$ if there is a run of $\mathcal{M}$ on input w that reaches $\widehat{q}$. ◀

To conclude from this lemma that the small term reachability problem is N2ExpTime-hard for TRSs whose termination can be shown using a polynomial order, it is enough to prove the following result.

▶ **Lemma 17.** *Termination of $R^{\mathcal{M}} \cup R_1 \cup R_2 \cup R_3$ can be shown using a polynomial order.*

Proof. Termination of $R^{\mathcal{M}} \cup R_1 \cup R_2 \cup R_3$ can be shown using the following polynomial interpretation of the function symbols:

- $a(x)$ is mapped to $x + 2$, for all tape symbols a of the TM (where a can also be the blank symbol b),
- $\#$ is mapped to 3,
- $q(x, y)$ is mapped to $x + y + 3$, for all states q of the TM, in particular also for q_0 and $\hat{q}$,
- $f(x)$ is mapped to $x + 2$,
- $+_1(x, y)$, $+_2(x, y)$, and $+_3(x, y)$ are mapped to $x + 2y + 1$,
- $d_1(x)$, $d_2(x)$, and $d_3(x)$ are mapped to $3x + 1$,
- $q_1(x)$, $q_2(x)$, and $q_3(x)$ are mapped to $3x^2 + 3x + 1$.

It remains to show that the polynomial order $\succ$ induced by this polynomial interpretation satisfies $g \succ d$ for all rules $g \to d$ of $R^{\mathcal{M}} \cup R_1 \cup R_2 \cup R_3$. First, we consider $R^{\mathcal{M}}$:

- for the rule $\hat{q}(x, y) \to \#$, the left-hand side is mapped to $x + y + 3$, and the right-hand side to 3, which is smaller that $x + y + 3$ for all instantiations of x, y with numbers > 0,

- for rules of the form $a(\hat{q}(x,y)) \to \hat{q}(x,y)$, the left-hand side is mapped to $x+y+5$, and the right-hand side to $x+y+3$,
- for all rules of $R^{\mathcal{M}}$ of the form $q(a(x), f(y)) \to a'(q'(x,y))$, the left-hand side is mapped to $(x+2)+(y+2)+3 = x+y+7$, and the right-hand side is mapped to $(x+y+3)+2 = x+y+5$,
- for all rules of $R^{\mathcal{M}}$ of the form $c(q(a(x), f(y))) \to q'(ca'(x), y)$, the left-hand side is mapped to $((x+2)+(y+2)+3)+2 = x+y+9$, and the right-hand side is mapped to $((x+2)+2)+y+3 = x+y+7$.

Next, we consider R_1:

- for the rule $q_0(y_1,y_2) +_1 q_0(z_1,z_2) \to q_0(y_1,y_2)$ of R_1, the left-hand side is mapped to $y_1+y_2+3+2(z_1+z_2+3)+1 = y_1+y_2+2z_1+2z_2+10$, and the right-hand side to y_1+y_2+3,
- for the rule $b(y) +_1 q_0(z_1,z_2) \to b(y)$ of R_1, the left-hand side is mapped to $y+2+2(z_1+z_2+3)+1 = y+2z_1+2z_2+9$, and the right-hand side to $y+2$,
- for the rule $x +_1 b(y) \to b(x +_1 y)$ of R_1, the left-hand side is mapped to $x+2(y+2)+1 = x+2y+5$, and the right-hand side to $(x+2y+1)+2 = x+2y+3$,
- for the rule $d_1(q_0(z_1,z_2)) \to q_0(z_1,z_2)$ of R_1, the left-hand side is mapped to $3(z_1+z_2+3)+1 = 3z_1+3z_2+10$, and the right-hand side to z_1+z_2+3,
- for the rule $d_1(b(x)) \to b(b(d_1(x)))$ of R_1, the left-hand side is mapped to $3(x+2)+1 = 3x+7$, and the right-hand side to $(3x+1+2)+2 = 3x+5$,
- for the rule $q_1(q_0(z_1,z_2)) \to q_0(z_1,z_2)$ of R_1, the left-hand side is mapped to $3(z_1+z_2+3)^2+3(z_1+z_2+3)+1$, and the right-hand side to z_1+z_2+3,
- for the rule $q_1(b(x)) \to q_1(x) +_1 b(d_1(x))$ of R_1, the left-hand side is mapped to $3(x+2)^2+3(x+2)+1 = 3x^2+15x+19$, and the right-hand side to $3x^2+3x+1+2(3x+1+2)+1 = 3x^2+9x+8$, as in Example 2.

The rules of R_2 and R_3 can be treated in a similar way. ◀

Combining the results obtained so far in this section, we thus have determined the exact complexity of the small term reachability problem for the class of TRSs considered here.

▶ **Theorem 18.** *The small term reachability problem for TRSs whose termination can be shown with a polynomial order is N2ExpTime-complete.*

In the setting considered in this section, restricting the attention to confluent TRSs does not reduce the complexity. Regarding the upper bound, the argument used in the proof of Proposition 7 does not apply since it is no longer the case that normal forms are of smallest size. Thus, one cannot reduce the complexity from N2ExpTime to 2ExpTime by only looking at a single rewrite sequence that ends in a normal form. However, our N2ExpTime-hardness proof does not directly work for confluent TRSs whose termination can be shown with a polynomial order. The reason is that, for a given nondeterministic Turing machine $\mathcal{M}$, the rewrite system $R^{\mathcal{M}} \cup R_1 \cup R_2 \cup R_3$ need not be confluent. In fact, for a given input word, there may be terminating runs of the TM that reach the accepting state $\hat{q}$, but also ones that do not reach this state. Using the former runs, our rewrite system can then generate the term #, whereas this is not possible if we use one of the latter runs.

We can, however, modify the system $R^{\mathcal{M}} \cup R_1 \cup R_2 \cup R_3$ such that it becomes confluent. To this end, we introduce two new function symbols $\#_1$ and $\#_0$ of arity 1 and 0, respectively. Moreover, we add the following rules R_c:

$$\begin{array}{rcll} g(x_1,\ldots,x_n) & \to & \#_1(\#_0) & \text{for all function symbols } g \text{ of arity} > 0 \text{ except } \#_1, \\ \#_1(\#_1(\#_0)) & \to & \#_1(\#_0), & \\ \# & \to & \#_1(\#_0). & \end{array}$$

Clearly, $R^{\mathcal{M}} \cup R_1 \cup R_2 \cup R_3 \cup R_c$ is confluent, because any term that is not in normal form (i.e., any term except variables, $\#_0$, $\#_1(\#_0)$, and terms of the form $\#_1(x)$ for variables x) has the only normal form $\#_1(\#_0)$ of size two. (This is the reason why we could not use a rule like $x +_1 q_0(z_1, z_2) \to x$ in R_1, because then $x +_1 q_0(z_1, z_2)$ would have the two normal forms x and $\#_1(\#_0)$.) However, the term $\#$ of size one is still only reachable from $t(w)$ if the final state of the TM is reached by a simulation of an accepting computation of $\mathcal{M}$. We extend the polynomial interpretation in the proof of Lemma 9 as follows:

- $\#_1(x)$ is mapped to $x + 1$,
- $\#_0$ is mapped to 1.

Then the polynomial order induced by this polynomial interpretation also orients the rules of R_c from left to right, i.e., termination of the resulting system can still be shown using a polynomial order.

▶ **Corollary 19.** *For confluent TRSs whose termination can be shown with a polynomial order, the small term reachability problem is N2ExpTime-complete.*

As shown in [16], if termination of a TRS can be shown with a *linear* polynomial order (i.e., where all polynomials have degree at most 1), then this implies an exponential bound on the lengths of reduction sequences. Again, this bound is tight and one can use the example showing this to obtain a TRS that generates an exponentially large tape and an exponentially large counter, similarly to what we have done in the general case.

▶ **Example 20.** Let R_d consist of just the two d-rules from Example 12. Then the term $d^\ell(s(0))$ can be reduced to $s^{2^\ell}(0)$.

▶ **Corollary 21.** *The small term reachability problem is NExpTime-complete for TRSs whose termination can be shown with a linear polynomial order. NExpTime-hardness already holds if only confluent systems are considered.*

Proof. The upper bound can be shown as before, i.e., one just needs to guess a reduction sequence (of exponential length) and then check the size of the obtained term.

For the lower bound, we proceed as in the proof of N2ExpTime-hardness for the case of general polynomial orders. Thus, we assume that $\mathcal{M}$ is an exponentially time bounded nondeterministic TM whose time-bound is $2^{p(\ell)}$ for a polynomial p, where ℓ is the length of the input word. Given such an input word $w = a_1 \dots a_\ell$ for $\mathcal{M}$, we now construct the term

$$t'(w) = d_1^{p(\ell)}(b(q_0(a_1 \dots a_\ell d_2^{p(\ell)}(b(\#)), d_3^{p(\ell)}(f(\#))))).$$

Instead of R_1, R_2, R_3, we now only need their rules for d_1, d_2, and d_3; let R'_d denote this system of 6 rules. As above, we can show that the term $t'(w)$ can be rewritten with $R^{\mathcal{M}} \cup R'_d$ to a term of size 1 iff $\mathcal{M}$ accepts the word w. Moreover, termination of $R^{\mathcal{M}} \cup R'_d$ can be proved by the linear polynomial order obtained from the one in the proof of Lemma 17 by removing the (non-linear) interpretations of q_1, q_2, q_3.

Similarly to the proof of Corollary 19, we can prove that NExpTime-hardness also holds for *confluent* TRSs whose termination can be shown with a linear polynomial order. The reason is that termination of the modified confluent TRS $R^{\mathcal{M}} \cup R'_d \cup R_c$ can be shown by the linear polynomial order that results from the one employed in the proof of Corollary 19 by removing the (non-linear) interpretations of q_1, q_2, q_3. ◀

6 Term rewriting systems shown terminating with a Knuth-Bendix order without special symbol

Without any restriction, there is no primitive recursive bound on the length of derivation chains for TRSs whose termination can be shown using a Knuth-Bendix order [16], but a uniform multiple recursive upper bound is shown in [15]. Here, we restrict the attention to KBOs without a *special symbol*, i.e., without a unary symbol of weight zero. For such KBOs, an exponential upper bound on the derivation length was shown in [16].[1] Given the results proven in the previous section, one could now conjecture that in this case the small term reachability problem is NExpTime-complete. However, we will show below that the complexity is actually only PSpace. In fact, the TRSs yielding the lower bounds for the derivation length considered in the previous section have not only long reduction chains (of double-exponential or exponential length), but are also able to produce large terms (of double-exponential or exponential size). For KBOs without special symbol, this is not the case. The following lemma provides us with a linear bound on the sizes of reachable terms. It will allow us to show a *PSpace upper bound* for the small term reachability problem.

▶ **Lemma 22.** *Let R be a TRS whose termination can be shown using a KBO without special symbol, and s_0, s_1 terms such that $s_0 \xrightarrow{*}_R s_1$. Then the size of s_1 is linearly bounded by the size of s_0, i.e., there is a constant c such that that $|s_1| \leq c{\cdot}|s_0|$ whenever $s_0 \xrightarrow{*}_R s_1$.*

Proof. Fix a KBO with weight function w showing termination of R such that all symbols of arity 1 have weight > 0. Let w_{min} be the minimal weight > 0 of a function symbol occurring in R or a variable,[2] i.e.,

$$w_{min} := \min\{w(f) \mid w(f) > 0 \text{ and } f \text{ is a function symbol in } R \text{ or a variable}\},$$

and let w_{max} be the maximal weight of a function symbol in R or a variable. As the weights of function symbols not occurring in R have no influence on the orientation of the rules in R with the given KBO, we can assume without loss of generality that their weight is w_{min}.

Let t be a term and $n_i(t)$ for $i = 0, \ldots, k$ the number of occurrences of symbols of arity i in t, where k is the maximal arity of a symbol occurring in t.[3] Note that $|t| = n_0(t) + n_1(t) + \ldots + n_k(t)$. The following fact, which can easily be shown by induction on the structure of t, is stated in [21]:

$$n_0(t) + n_1(t) + \ldots + n_k(t) = 1 + 1{\cdot}n_1(t) + 2{\cdot}n_2(t) + \ldots + k{\cdot}n_k(t).$$

In particular, this implies that $n_0(t) \geq n_2(t) + \ldots + n_k(t)$. Since symbols of arity 0 and 1 have weights > 0, we know that

$$w(t) \geq w_{min}{\cdot}(n_0(t) + n_1(t)) \geq w_{min}{\cdot}n_0(t) \geq w_{min}{\cdot}(n_2(t) + \ldots + n_k(t)).$$

Consequently, $2{\cdot}w_{min}^{-1}{\cdot}w(t) \geq n_0(t) + n_1(t) + \ldots + n_k(t) = |t|$. This shows that the size of a term is linearly bounded by its weight. Conversely, it is easy to see that the weight of a term is linearly bounded by its size: $w(t) \leq w_{max}{\cdot}|t|$.

Now, assume that $s_0 \xrightarrow{*}_R s_1$. Since termination of R is shown with our given KBO, we know that $w(s_0) \geq w(s_1)$, and thus $w_{max}{\cdot}|s_0| \geq w(s_1) \geq 1/2{\cdot}w_{min}{\cdot}|s_1|$. This yields $|s_1| \leq 2{\cdot}w_{min}^{-1}{\cdot}w_{max}{\cdot}|s_0|$. ◀

[1] Actually, this result was shown in [16] only for KBOs using weights in $\mathbb{N}$, but it also holds for KBOs with non-negative weights in $\mathbb{R}$. This is an easy consequence of our Lemma 22.
[2] Recall that all variables have the same weight $w_0 > 0$.
[3] Variables have arity 0.

In particular, this means that the terms encountered during a rewriting sequence starting with a term s can each be stored using only polynomial space in the size of s. Given that the length of such a sequence is exponentially bounded, we can decide the small term reachability problem by the following NPSpace algorithm:

- guess a rewrite sequence $s \to_R s_1 \to_R s_2 \to_R \dots$ and always store only the current term;
- in each step, check whether $|s_i| \leq n$ holds. If the answer is "yes" then stop and accept. Otherwise, guess the next rewriting step; if this is not possible since s_i is irreducible, then stop and reject.

This algorithm needs only polynomial space since, by Lemma 22, the size of each term s_i is linearly bounded by the size of s. It always terminates since R is terminating. If there is a term of size $\leq n$ reachable from s, then the algorithm is able to guess the sequence leading to it, and thus it has an accepting run. Otherwise, all runs are terminating and rejecting. Since, by Savitch's theorem [27], NPSpace = PSpace, we obtain the following complexity upper bound.

▶ **Lemma 23.** *The small term reachability problem is in PSpace for TRSs whose termination can be shown with a KBO without special symbol.*

It remains to prove the corresponding *lower bound*. Let $\mathcal{M}$ be a polynomial space bounded TM, and p the polynomial that yields the space bound. Then there is a polynomial q such that any run of $\mathcal{M}$ longer than $2^{q(\ell)}$ on an input word w of length ℓ is cyclic. Thus, to check whether $\mathcal{M}$ accepts w, it is sufficient to consider only runs of length at most $2^{q(\ell)}$. However, in contrast to the reduction used in the previous section, we cannot generate an exponentially large unary down counter using a TRS whose termination can be shown with a KBO without special symbol. Instead, we use a polynomially large *binary* down counter that is decremented, starting with the binary representation $10^{q(\ell)}$ of $2^{q(\ell)}$ (see Example 3). For example, if $q(\ell) = 3$, then we represent the number $2^{q(\ell)} = 2^3 = 8$ as the binary number $10^{q(\ell)} = 1000$. The construction of the TRS $R^{\mathcal{M}}_{bin}$ simulating $\mathcal{M}$ given below is very similar to the construction given in the proof of Lemma 7 in [10].

As signature for $R^{\mathcal{M}}_{bin}$ we again use the tape symbols of $\mathcal{M}$ as unary function symbols, but now also the states are treated as unary symbols. In addition, we need the unary function symbols 0 and 1 to represent the counter, as well as primed versions $a', q', 1'$ of the tape symbols a, the states q, and the symbol 1. For a given input word $w = a_1 \dots a_\ell$ of $\mathcal{M}$, we now construct a term that starts with the binary representation of $2^{q(\ell)}$ and is followed by enough tape space for a $p(\ell)$ space bounded TM to work on:

$$t(w) := 10^{q(\ell)}(b^{p(\ell)}(q_0(a_1 \dots a_\ell(b^{p(\ell)-\ell}(\#))))).$$

Clearly, $t(w)$ can be constructed in polynomial time.

The TRS $R^{\mathcal{M}}_{bin}$ is now constructed as follows. The first part decrements the counter (as in Example 3) and by doing so "sends a prime" to the right:

$$\begin{array}{llll} 1(a(x)) \to 0(a'(x)) & \text{and} & 0(a(x)) \to 1'(a'(x)) & \text{for all tape symbols } a, \\ 0(1'(x)) \to 1'(1(x)), & & 1(1'(x)) \to 0(1(x)). & \end{array}$$

The prime can go to the right on the tape until it reaches a state, which it then turns into its primed version:

$$a'g(x) \to ag'(x) \qquad \text{for tape symbols } a \text{ and tape symbols or states } g.$$

Only primed states can perform a transition of the TM:

$$\begin{array}{ll} q_1'(a_1(x)) \to a_2(q_2(x)) & \text{for each transition } (q_1, a_1, q_2, a_2, r) \text{ of } \mathcal{M}, \\ c(q_1'(a_1(x))) \to q_2(c(a_2(x))) & \text{for each transition } (q_1, a_1, q_2, a_2, l) \text{ of } \mathcal{M} \\ & \text{and tape symbol } c. \end{array}$$

Again, the blank symbol b is also considered as a tape symbol of $\mathcal{M}$. Note that the rôle of the counter is not to restrict the number of transition steps simulated by $R_{bin}^{\mathcal{M}}$. Instead it produces enough primes to allow the simulation of at least $2^{q(\ell)}$ steps, while termination can still be shown using a KBO without special symbol.

Once the unique final accepting state $\hat{q}$ is reached, we remove all symbols other than #:

$$\begin{array}{ll} a(\hat{q}(x)) \to \hat{q}(x) & \text{where } a \text{ is a tape symbol or 0 or 1,} \\ \hat{q}(x) \to \#. & \end{array}$$

▶ **Lemma 24.** *The term $t(w)$ can be rewritten with $R_{bin}^{\mathcal{M}}$ to a term of size 1 iff $\mathcal{M}$ accepts the word w.*

Proof. If $\mathcal{M}$ accepts the word w, then there is a run of $\mathcal{M}$ on input w that ends in the state $\hat{q}$, uses at most $p(\ell)$ space, and requires at most $2^{q(\ell)}$ steps. This run can be simulated by $R_{bin}^{\mathcal{M}}$ by decrementing the counter, sending a prime to the state, applying a transition, decrementing the counter, etc. Since the counter can be decremented $2^{q(\ell)}$ times, we can use this approach to simulate a run of length at most $2^{q(\ell)}$. Once the accepting state is reached, we can use the last two rules to reach the term #, which has size 1.

Conversely, we can only reach a term of size one, if these cancellation rules are applied. This is only possible if first the accepting state has been reached by simulating an accepting run of $\mathcal{M}$. ◀

To conclude from this lemma that the small term reachability problem is PSpace-hard for TRSs whose termination can be shown using a KBO without special symbol, it is enough to show the following result.

▶ **Lemma 25.** *Termination of $R_{bin}^{\mathcal{M}}$ can be shown with a KBO without special symbol.*

Proof. It is easy to see that the KBO that assigns weight 1 to all function symbols and to all variables, and uses the precedence order $1 > 0 > 1'$ and $q' > a' > a > q$ for states q and tape symbols a, orients all rules of $R_{bin}^{\mathcal{M}}$ from left to right.[4] ◀

Combining the results obtained so far in this section, we thus have determined the exact complexity of the small term reachability problem for our class of TRSs.

▶ **Theorem 26.** *The small term reachability problem is PSpace-complete for TRSs whose termination can be shown with a KBO without special symbol.*

As in the case of the TRSs considered in the previous section, confluence does not reduce the complexity of the small term reachability problem for TRSs shown terminating with a KBO without special symbol. In fact, we can again extend the TRS $R_{bin}^{\mathcal{M}}$ such that it becomes confluent. To this purpose, we add two new function symbols $\#_1$ and $\#_0$ of respective arity 1 and 0, and two new rules:

$$\begin{array}{ll} g(x) \to \#_1(\#_0) & \text{for all unary function symbols } g \text{ different from } \#_1, \\ \# \to \#_1(\#_0). & \end{array}$$

[4] This KBO is similar to the one introduced in Example 10 of [10].

With this addition, every non-variable term built using the original signature of $R^{\mathcal{M}}_{bin}$ can be reduced to $\#_1(\#_0)$, which shows confluence. To show termination of the extended TRS, we modify and extend the KBO from the proof of Lemma 25 as follows. All function symbols in the original signature of $R^{\mathcal{M}}_{bin}$ (including $\#$) now get weight 2, and the symbols $\#_1$ and $\#_0$ as well as the variables get weight 1. The precedence order is extended by setting $g > \#_1$ for all function symbols g in the original signature of $R^{\mathcal{M}}_{bin}$. It is easy to see that the KBO defined this way shows that the extended TRS is terminating.

▶ **Corollary 27.** *For confluent TRSs whose termination can be shown with a KBO without special symbol, the small term reachability problem is PSpace-complete.*

7 Conclusion

The results of this paper show that the complexity of the small term reachability problem is closely related to the derivational complexity of the class of term rewriting systems considered. Interestingly, restricting the attention to confluent TRSs reduces the complexity only for the class of length-reducing systems, but not for the other two classes considered in this paper. The investigations in this paper were restricted to classes of TRSs defined by reduction orders (restricted form of KBO and polynomial orders) that yield relatively low bounds on the derivational complexity of the TRS. The derivational complexity of TRSs shown terminating by KBOs with a unary function symbol of weight zero or by recursive path orders is much higher [14, 15, 23, 24, 29]. From a theoretical point of view, it would be interesting to see whether using such reduction orders or other more powerful techniques [13] for showing termination also results in a very high complexity of the small term reachability problem. In fact, as we have seen in this paper, the complexity of this problem not only depends on the length of reduction sequences, but also on whether one can use long sequences to generate large terms.

On the practical side, up to now we have only used length-reducing rules to shorten DL proofs. Basically, these rules are generated by finding frequent proof patterns (currently by hand) and replacing them by a new "macro rule". The results of Section 4 show that, in this case, confluence of the rewrite system is helpful. When translating between different proof calculi, length-reducing systems will probably not be sufficient. Therefore, we will investigate with what kinds of techniques proof rewriting systems (e.g., translating between different proof calculi for $\mathcal{EL}$) can be shown terminating. Are polynomial orders or KBOs without unary function symbol of weight zero sufficient, or are more powerful approaches for showing termination needed? In this context, it might also be interesting to consider rewriting modulo equational theories [8, 18] and associated approaches for showing termination [1, 11, 19, 26]. For example, it makes sense not to distinguish between proof steps that differ only in the order of the prerequisites. Hence, rewriting such proofs could be represented via term rewriting modulo associativity and commutativity.

References

1 Beatriz Alarcón, Salvador Lucas, and José Meseguer. A dependency pair framework for $A \vee C$-termination. In Peter Csaba Ölveczky, editor, *Rewriting Logic and Its Applications - 8th International Workshop, WRLA 2010, Revised Selected Papers*, volume 6381 of *Lecture Notes in Computer Science*, pages 35–51. Springer, 2010. doi:10.1007/978-3-642-16310-4_4.

2 Christian Alrabbaa, Franz Baader, Stefan Borgwardt, Raimund Dachselt, Patrick Koopmann, and Julián Méndez. Evonne: Interactive proof visualization for description logics (system description). In Jasmin Blanchette, Laura Kovács, and Dirk Pattinson, editors, *Automated Reasoning - 11th International Joint Conference, IJCAR 2022, Proceedings*, volume 13385 of *Lecture Notes in Computer Science*, pages 271–280. Springer, 2022. doi:10.1007/978-3-031-10769-6_16.

3 Christian Alrabbaa, Franz Baader, Stefan Borgwardt, Patrick Koopmann, and Alisa Kovtunova. Finding small proofs for description logic entailments: Theory and practice. In Elvira Albert and Laura Kovács, editors, *LPAR 2020: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Proceedings*, volume 73 of *EPiC Series in Computing*, pages 32–67. EasyChair, 2020. doi:10.29007/NHPP.

4 Christian Alrabbaa, Franz Baader, Stefan Borgwardt, Patrick Koopmann, and Alisa Kovtunova. Finding good proofs for description logic entailments using recursive quality measures. In André Platzer and Geoff Sutcliffe, editors, *Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, Proceedings*, volume 12699 of *Lecture Notes in Computer Science*, pages 291–308. Springer, 2021. doi:10.1007/978-3-030-79876-5_17.

5 Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the $\mathcal{EL}$ envelope. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 364–369. Professional Book Center, 2005. URL: http://ijcai.org/Proceedings/05/Papers/0372.pdf.

6 Franz Baader, Ian Horrocks, Carsten Lutz, and Ulrike Sattler. *An Introduction to Description Logic*. Cambridge University Press, 2017.

7 Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

8 Leo Bachmair and Nachum Dershowitz. Completion for rewriting modulo a congruence. *Theor. Comput. Sci.*, 67(2&3):173–201, 1989. doi:10.1016/0304-3975(89)90003-0.

9 Guillaume Bonfante, Adam Cichon, Jean-Yves Marion, and Hélène Touzet. Algorithms with polynomial interpretation termination proof. *J. Funct. Program.*, 11(1):33–53, 2001. doi:10.1017/S0956796800003877.

10 Guillaume Bonfante and Georg Moser. Characterising space complexity classes via Knuth-Bendix orders. In Christian G. Fermüller and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 17th International Conference, LPAR-17, Proceedings*, volume 6397 of *Lecture Notes in Computer Science*, pages 142–156. Springer, 2010. doi:10.1007/978-3-642-16242-8_11.

11 Jürgen Giesl and Deepak Kapur. Dependency pairs for equational rewriting. In Aart Middeldorp, editor, *Rewriting Techniques and Applications, 12th International Conference, RTA 2001, Proceedings*, volume 2051 of *Lecture Notes in Computer Science*, pages 93–108. Springer, 2001. doi:10.1007/3-540-45127-7_9.

12 Jürgen Giesl, Matthias Raffelsieper, Peter Schneider-Kamp, Stephan Swiderski, and René Thiemann. Automated termination proofs for Haskell by term rewriting. *ACM Trans. Program. Lang. Syst.*, 33(2):7:1–7:39, 2011. doi:10.1145/1890028.1890030.

13 Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, and Stephan Falke. Mechanizing and improving dependency pairs. *J. Autom. Reason.*, 37(3):155–203, 2006. doi:10.1007/S10817-006-9057-7.

14 Dieter Hofbauer. Termination proofs by multiset path orderings imply primitive recursive derivation lengths. *Theor. Comput. Sci.*, 105(1):129–140, 1992. doi:10.1016/0304-3975(92)90289-R.

15 Dieter Hofbauer. An upper bound on the derivational complexity of Knuth-Bendix orderings. *Inf. Comput.*, 183(1):43–56, 2003. doi:10.1016/S0890-5401(03)00008-7.

16 Dieter Hofbauer and Clemens Lautemann. Termination proofs and the length of derivations (preliminary version). In Nachum Dershowitz, editor, *Rewriting Techniques and Applications, 3rd International Conference, RTA-89, Proceedings*, volume 355 of *Lecture Notes in Computer Science*, pages 167–177. Springer, 1989. doi:10.1007/3-540-51081-8_107.

17 Gérard Huet and Dallas S. Lankford. On the uniform halting problem for term rewriting systems. INRIA Rapport de Recherche No. 283, 1978. URL: https://www.ens-lyon.fr/LIP/REWRITING/TERMINATION/Huet_Lankford.pdf.

18 Jean-Pierre Jouannaud and Hélène Kirchner. Completion of a set of rules modulo a set of equations. *SIAM J. Comput.*, 15(4):1155–1194, 1986. doi:10.1137/0215084.

19 Jean-Pierre Jouannaud and Claude 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.

20 Yevgeny Kazakov, Markus Krötzsch, and Frantisek Simancik. The incredible ELK - from polynomial procedures to efficient reasoning with $\mathcal{EL}$ ontologies. *J. Autom. Reason.*, 53(1):1–61, 2014. doi:10.1007/S10817-013-9296-3.

21 Donald E. Knuth and Peter B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*. Pergamon Press, Oxford, 1970.

22 Dallas S. Lankford. On proving term rewriting systems are Noetherian. Memo MTP-3, Math. Dept., Louisiana Technical University, Ruston, LA, 1979. URL: http://www.ens-lyon.fr/LIP/REWRITING/TERMINATION/Lankford_Poly_Term.pdf.

23 Ingo Lepper. Derivation lengths and order types of Knuth-Bendix orders. *Theor. Comput. Sci.*, 269(1-2):433–450, 2001. doi:10.1016/S0304-3975(01)00015-9.

24 Ingo Lepper. Simply terminating rewrite systems with long derivations. *Arch. Math. Log.*, 43(1):1–18, 2004. doi:10.1007/S00153-003-0190-2.

25 Robert Nieuwenhuis and Albert Rubio. Paramodulation-based theorem proving. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning, Vol. I*, pages 371–443. Elsevier and MIT Press, 2001. doi:10.1016/B978-044450813-3/50009-6.

26 Albert Rubio. A fully syntactic AC-RPO. *Inf. Comput.*, 178(2):515–533, 2002. doi:10.1006/INCO.2002.3158.

27 Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970. doi:10.1016/S0022-0000(70)80006-X.

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

29 Andreas Weiermann. Termination proofs for term rewriting systems by lexicographic path orderings imply multiply recursive derivation lengths. *Theor. Comput. Sci.*, 139(1&2):355–362, 1995. doi:10.1016/0304-3975(94)00135-6.

A Categorical Approach to DIBI Models

Tao Gu
University College London, UK

Jialu Bao
Cornell University, Ithaca, NY, USA

Justin Hsu
Cornell University, Ithaca, NY, USA

Alexandra Silva
Cornell University, Ithaca, NY, USA

Fabio Zanasi
University College London, UK
University of Bologna, OLAS team (INRIA), Italy

Abstract

The logic of Dependence and Independence Bunched Implications (DIBI) is a logic to reason about conditional independence (CI); for instance, DIBI formulas can characterise CI in discrete probability distributions and in relational databases, using a probabilistic DIBI model and a similarly-constructed relational model. Despite the similarity of the two models, there lacks a uniform account. As a result, the laborious case-by-case verification of the frame conditions required for constructing new models hinders them from generalising the results to CI in other useful models such that continuous distribution. In this paper, we develop an abstract framework for systematically constructing DIBI models, using category theory as the unifying mathematical language. We show that DIBI models arise from arbitrary symmetric monoidal categories with copy-discard structure. In particular, we use string diagrams – a graphical presentation of monoidal categories – to give a uniform definition of the parallel composition and subkernel relation in DIBI models. Our approach not only generalises known models, but also yields new models of interest and reduces properties of DIBI models to structures in the underlying categories. Furthermore, our categorical framework enables a comparison between string diagrammatic approaches to CI in the literature and a logical notion of CI, defined in terms of the satisfaction of specific DIBI formulas. We show that the logical notion is an extension of string diagrammatic CI under reasonable conditions.

2012 ACM Subject Classification Theory of computation → Logic; Theory of computation → Semantics and reasoning; Theory of computation → Models of computation

Keywords and phrases Conditional Independence, Dependence Independence Bunched Implications, String Diagrams, Markov Categories

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.17

Funding *Tao Gu*: Tao Gu is supported by the UK EPSRC grant EP/S013008/1.
Justin Hsu: Justin Hsu is partially supported by NSF grant #194313.
Alexandra Silva: Silva was partially supported by ERC grant Autoprobe (no. 101002697).
Fabio Zanasi: Fabio Zanasi acknowledges support from EPSRC EP/V002376/1 and MIUR P2022HXNSC (PRIN 2022 PNRR – Next Generation EU).

Acknowledgements We thank the anonymous reviewers for their close reading and detailed feedback.

1 Introduction

Conditional independence (CI) is a fundamental concept that can be traced back to the pioneer work on probabilities in Bayes [6] and Laplace [25]. In modern days, this notion is formalised and applied across various fields of science. For instance, CI is a central concept

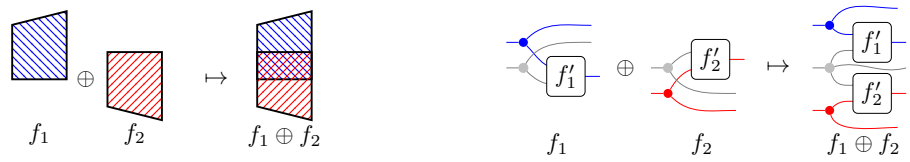
in the first formal definition of secrecy by Shannon [37], and remains crucial in various subsequent works in cryptography [26, 30]; graphical models leverage CI relations to have efficient representations of probabilistic phenomenons [32, 21, 13]. The core idea of CI is straightforward: events A and B are "independent" when information about one does not convey information about the other; events A and B are "conditionally independent" given event C if, with knowledge of event C, events A and B become independent. Albeit intuitive, reasoning about conditional independence is intricate, leading to extensive research aimed at formalising such reasoning [32, 14].

For probabilistic programs, an extension of standard programs with constructs to sample from distributions, formal methods for (conditional) independence have emerged as powerful tools for program verification. For instance, Barthe et al. [5] introduced Probabilistic Separation Logic (PSL) to formalise several cryptography protocols, where the independence of variables guarantees no leakage of information and thus security of the algorithms. A follow-up work from Bao et al. [4] proposed the logic of *Dependence and Independence Bunched Implications* (DIBI), which enhances PSL with the ability to reason about *conditional* independence. Syntactically, DIBI extends the logic of Bunched Implications (BI) [28, 34], which is the assertion logic underpinning Separation Logic (SL) [35] and PSL, with a non-commutative conjunction $\fatsemi$ and its adjoints. Semantically, as in BI, the separating conjunction $*$ is interpreted through a partial operation $\oplus$ on states, regarded as the parallel composition. In addition, they define a sequential composition $\odot$ to interpret $P \fatsemi Q$. Informally, $P * Q$ says that P and Q hold in states that can be separated, and $P \fatsemi Q$ expresses a possible dependency of Q on P. Section 3 will review the logic in more details.

Bao et al. [4] introduced two kinds of semantic models for DIBI logic: the probabilistic DIBI models for reasoning about CI of variables in discrete probabilistic computation, and the relational DIBI models for expressing the CI notion in relational databases called join dependency. These two models are defined analogously, yielding similar conditions for one to laboriously check to ensure that they are models. Such similarity led the authors to conjecture a family of categorical DIBI models that induce these concrete models as instances.

We believe that such categorical models would facilitate the construction of new models and set out to solve the conjecture with a simple observation: in both the probabilistic and relational DIBI models, the states resemble *Markov kernels* – maps from input elements to distributions/powersets over output elements. Such DIBI states can be identified categorically as morphisms in the Kleisli categories associated to the discrete distribution monad $\mathcal{D}$ (Definition 36) or the nonempty powerset monad $\mathcal{P}_i$ (Definition 37). However, giving a categorical definition for the parallel compositions $\oplus$ is difficult. The previous work [4] gives Figure 1a as a pictorial intuition for the parallel composition. The states are drawn as trapezoids, with the short and long vertical sides representing the input and output domains, respectively. There, given a blue map f_1 and a red map f_2, their parallel composition $f_1 \oplus f_2$ takes as input the union of their inputs. Then, each f_i takes its counterpart in the combined input domain and generates an output. Finally these two outputs are combined to be the output of $f_1 \oplus f_2$. This parallel composition *is partial* because the combination of their outputs is allowed only when the variables overlap in particular ways. This creates a challenge to capture DIBI models categorically because, in a categorical setting, the domains and codomains of DIBI states are objects, and it is not obvious how to define the overlap of objects.

Our solution stems from a formalisation of this graphical intuition through *string diagrams*, a pictorial formalism for monoidal categories. String diagrams are widely adopted as intuitive yet mathematically rigorous reasoning tools across different areas of science, see [33] for



(a) Parallel composition depicted in [4]. **(b)** A diagrammatic representation.

Figure 1 Intuition for parallel composition.

an overview. We formalise the trapezoids intuition in Figure 1a into string diagrams in Figure 1b. The maps previously embodied as trapezoids now have a fork shape, with some branches being straight lines and some other branches going through boxes. The boxes represent arbitrary morphisms in the underlying category, and the straight lines represent the identity morphisms. Whereas composition of two DIBI states were hand-waved as two trapezoids tiled together in Figure 1a, the string diagram defines it precisely: the overlap of the two trapezoids is witnessed by the grey wires, and the composition joins two diagrams side-by-side with the grey wires shared. We show in Section 4 that this string diagram representation yields DIBI models in any category with enough structure to interpret $\multimap\!\!\subset$, namely, Markov categories [9, 12]. We then derive existing and new concrete DIBI models as instances in Section 5.

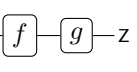
This framework also enables a comparison between different characterisations of conditional independence (CI). While Bao et al. [4] show that probabilistic or relational CI are both captured by some DIBI formulas, it is unclear if these formulas generalise to CI in other models and how they compare to other abstract notions of CI. Since we can construct categorical DIBI models based on any Markov categories, we define a logical notion of CI for morphisms in Markov categories as satisfaction of those DIBI formulas. In Section 6, we investigate the relationship between our "logical" CI and various CI notions based on categorical structures from literature in synthetic statistics [9, 12] and identify the conditions that make them equivalent.

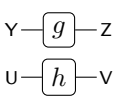
Throughout the paper we fix a countably infinite set of variables Var, use $x, y, z, \ldots$ for elements of Var, and use $W, X, Y, \ldots$ for finite subsets of Var.

2 Category Theory Preliminaries

Unless specified, all monoidal categories we consider are strict and we write $dom(f)$ and $cod(f)$ for the domain and codomain of any morphism f. We write $\langle \mathbb{C}, \otimes, \mathsf{I} \rangle$ for a (strict) monoidal category, where $\otimes$ is the monoidal product and I the unit object of $\mathbb{C}$. If it is also symmetric, we write $\sigma_{\mathsf{A},\mathsf{B}} \colon \mathsf{A} \otimes \mathsf{B} \to \mathsf{B} \otimes \mathsf{A}$ for the symmetry natural transformation indexed by objects A and B.

As detailed for instance in [36, 33, 11], morphisms of symmetric monoidal categories have a graphical presentation as string diagrams, where sequential composition and monoidal product are depicted as concatenation and juxtaposition of diagrams, respectively: given morphisms $f \colon \mathsf{X} \to \mathsf{Y}$, $g \colon \mathsf{Y} \to \mathsf{Z}$, $h \colon \mathsf{U} \to \mathsf{V}$,

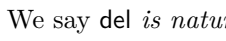
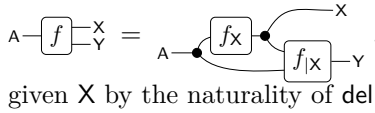
$g \circ f =$ 

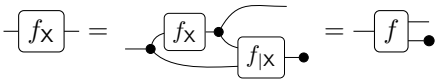
$\qquad g \otimes h =$ 

We read string diagrams from left to right, and tensor products from top to bottom. Object labels in the diagrams are omitted when they are evident or irrelevant to the context. Symmetries are indicated with the string diagram  . We call string diagrams consisting solely of combinations of s *rewirings*: intuitively, they permute the order of the objects.

We use the notion of a Markov category, which suitably generalises categories of probabilistic processes [12]. First, a *copy-delete category* (*CD category*) is a symmetric monoidal category (SMC) $\langle \mathbb{C}, \otimes, \mathsf{I} \rangle$ with "copy" $\mathsf{copy}_{\mathsf{C}}$ and "delete" $\mathsf{del}_{\mathsf{C}}$ morphisms for each object C, drawn diagrammatically as  and respectively, that form a commutative comonoid:

Because of the leftmost equation above, we sometimes write a "trident"  for either side of it. Moreover, both copy and del need to be compatible with the monoidal structure:

We say del *is natural* if  for every morphism f. A *Markov category* is a CD category in which del is natural. A CD category $\mathbb{C}$ *has conditionals* if for each morphism $f\colon \mathsf{A} \to \mathsf{X}\otimes\mathsf{Y}$, there exist (not necessarily unique) morphisms $f_{\mathsf{X}}\colon \mathsf{A} \to \mathsf{X}$ (called the *marginal*) and $f_{|\mathsf{X}}\colon \mathsf{X} \to \mathsf{Y}$ (called the *conditional*) such that  . When $\mathbb{C}$ is a Markov category, such marginal f_{X} is unique given X by the naturality of del:



3 DIBI Logic and its Probabilistic Model

In this section we review the logic of *Dependence and Independence Bunched Implications* (DIBI). For space reasons, we focus on the discrete probabilistic model for DIBI. Interested readers may refer to [4] for the relational model, whose construction follows similar steps.

DIBI formulas (based on a set $\mathcal{AP}$ of atomic formulas) are defined inductively as follows:

$$P, Q ::= p \in \mathcal{AP} \mid \top \mid I \mid P \wedge Q \mid P \to Q \mid P * Q \mid P \mathrel{-\!\!*} Q \mid P \fatsemi Q \mid P \multimap Q \mid P \mathrel{\circ\!\!-} Q$$

The additive conjunction $\wedge$ is the standard Boolean conjunction. The multiplicative conjunction $*$ states that P and Q are independent. Both are already present in BI. DIBI extends BI with the non-commutative conjunction $\fatsemi$[1], where $P \fatsemi Q$ states that Q may depend on P. The operation $\mathrel{-\!\!*}$ is adjoint to $*$, $\to$ is adjoint to $\wedge$, and $\multimap$, $\mathrel{\circ\!\!-}$ are adjoints to $\fatsemi$. DIBI formulas are interpreted on DIBI *models*, each consisting of a *DIBI frame* on a set of states A and a *valuation* function $\mathcal{V}\colon \mathcal{AP} \to \mathcal{P}(A)$ that maps an atomic proposition to the set of states on which it is true. While a BI frame is based on a partial commutative monoid [10], a DIBI frame consists of two monoids (one commutative and one not) on the same underlying set, taking care of the two non-additive conjunctions $*$ and $\fatsemi$, respectively.

[1] Not to be confused with the additive context constructor which is also denoted as $\fatsemi$ in the standard BI literature such as [28, 34].

$a \oplus b \doteq b \oplus a$	($\oplus$-Com)	$\exists e \in E\colon a = e \odot a$	($\odot$-UnitExist$_\mathrm{L}$)
$\exists e \in E\colon a = e \oplus a$	($\oplus$-UnitExist)	$\exists e \in E\colon a = a \odot e$	($\odot$-UnitExist$_\mathrm{R}$)
$(a \oplus b) \oplus c \doteq a \oplus (b \oplus c)$	($\oplus$-Assoc)	$(a \odot b) \odot c \doteq a \odot (b \odot c)$	($\odot$-Assoc)
$e \in E \,\&\, (a \oplus e)\!\downarrow \implies (a \oplus e) \sqsupseteq a$			($\oplus$-UnitCoh)
$e \in E \,\&\, (a \odot e)\!\downarrow \implies (a \odot e) \sqsupseteq a$			($\odot$-UnitCoh$_\mathrm{R}$)
$e \in E \,\&\, e' \sqsupseteq e \implies e' \in E$			(UnitClosure)
$(a \oplus b)\!\downarrow \,\&\; a \sqsupseteq a' \,\&\, b \sqsupseteq b' \implies (a' \oplus b')\!\downarrow \,\&\, (a \oplus b) \sqsupseteq (a' \oplus b')$			($\oplus$-DownClosed)
$(a \odot b)\!\downarrow \,\&\, (a \odot b) \sqsubseteq c' \implies \exists a', b'\colon a' \sqsupseteq a \,\&\, b' \sqsupseteq b \,\&\, c' = (a' \odot b')$			($\odot$-UpClosed)
$(a_1 \odot a_2) \oplus (b_1 \odot b_2) \doteq (a_1 \oplus b_1) \odot (a_2 \oplus b_2)$			(RevExchange)

Figure 2 DIBI frame conditions (with implicit outermost universal quantifiers), where $\downarrow$ stands for "is defined", $\doteq$ means "equal when either side is defined".

▶ **Definition 1** ([4]). *A* DIBI frame *is a tuple $\mathcal{A} = \langle A, \sqsubseteq, \oplus, \odot, E\rangle$, where A is a set of states, $\sqsubseteq$ is a preorder on A, $E \subseteq A$ are units, and $\oplus, \odot\colon A \times A \rightharpoonup A$ are partial binary operations*[2]*, satisfying the frame conditions in Figure 2.*

The operations $\odot$ and $\oplus$ are referred to as the sequential and parallel compositions of states. Intuitively, $a \sqsubseteq b$ says that a can be extended to b, and E is the set of states that act as units for these operations. For capturing conditional independence, atomic propositions $\mathcal{AP}$ have the form $S \triangleright [T]$, for finite sets of variables S, T. Roughly, $S \triangleright [T]$ means the values of variables in T only depend on that of S. We now present the semantics of DIBI formulas, restricting to the fragment needed for the current work.

▶ **Definition 2.** *Given a DIBI model $\langle \mathcal{A}, \mathcal{V}\rangle$,* satisfaction $\vDash_{\mathcal{V}}$ *of DIBI$_{\{\wedge,*,\mathbin{⫽}\}}$-formulas at $\mathcal{A}$-states is inductively defined as follows:*

$a \vDash_{\mathcal{V}} I$ *iff* $a \in E$		$a \vDash_{\mathcal{V}} \top$ *always*
$a \vDash_{\mathcal{V}} (A \triangleright [B])$	*iff*	$a \in \mathcal{V}(A \triangleright [B])$
$a \vDash_{\mathcal{V}} P \wedge Q$	*iff*	$a \vDash_{\mathcal{V}} P$ *and* $a \vDash_{\mathcal{V}} Q$
$a \vDash_{\mathcal{V}} P * Q$	*iff*	$\exists b_1, b_2 \in A$ *such that* $b_1 \oplus b_2 \sqsubseteq a$, $b_1 \vDash_{\mathcal{V}} P$, $b_2 \vDash_{\mathcal{V}} Q$
$a \vDash_{\mathcal{V}} P \mathbin{⫽} Q$	*iff*	$\exists b_1, b_2 \in A$ *such that* $b_1 \odot b_2 = a$, $b_1 \vDash_{\mathcal{V}} P$, $b_2 \vDash_{\mathcal{V}} Q$

For a concrete example of DIBI models, we review the probabilistic models on program memories. Let Val be a set of values, to which variables in Var are assigned. A *memory over* a finite set of variables X is a function $\mathbf{m}\colon X \to \mathrm{Val}$, and the *memory space* over X is the set of all memories over X, denoted as $\mathbf{M}[X; \mathrm{Val}]$, or $\mathbf{M}[X]$ when Val is clear. Given a memory $\mathbf{m} \in \mathbf{M}[X]$ and a subset $U \subseteq X$, the memory $\mathbf{m}^U\colon U \to \mathrm{Val}$ is the restriction of $\mathbf{m}$ to the domain U. To express probabilistic features, we use $\mathcal{D}S$ to denote the set of discrete distributions over S; that is, the set of all $\mu\colon S \to [0,1]$ such that the *support* $\mathrm{supp}(\mu) = \{s \in S \mid \mu(s) > 0\}$ is finite, and $\sum_{s\in S} \mu(s) = 1$. A dirac distribution δ_s on an outcome s is the distribution such that $\delta_s(s) = 1$, and $\delta_s(s') = 0$ for any $s' \neq s$. Given a distribution μ in $\mathcal{D}\mathbf{M}[X]$, if $Y \subseteq X$, we define the marginalisation of μ to $\mathcal{D}\mathbf{M}[Y]$, written as $\pi_Y \mu$, by letting $(\pi_Y \mu)(\mathbf{m}') = \sum_{\mathbf{m}\in\mathbf{M}[X] \mid \mathbf{m}^Y = \mathbf{m}'} \mu(\mathbf{m})$.

[2] Note that, even though $\odot, \oplus$ are also partial in the models considered in [4], they have type $A \times A \to \mathcal{P}(A)$ in that work. This is because the authors obtain completeness of DIBI logic using a method developed by Docherty [10], which only works for the more general type. Because the operations are actually partial rather than non-deterministic, and we are not interested in completeness here, we stick to the more accurate type.

We are now ready to introduce the notion of *probabilistic input-preserving kernels*. In words, a probabilistic kernel f maps a memory $\mathbf{m}$ on X to a distribution of memories on $Y \supseteq X$ whose support contains only memories $\mathbf{m}'$ that faithfully extend $\mathbf{m}$ (thus the name "input-preserving"). Alternatively, f can be seen as a conditional distribution $\Pr(Y \mid X)$ where $Y \supseteq X$, such that $\Pr(Y = B \mid X = A)$ is nonzero only if B restricted to X equals A.

▶ **Definition 3** ([4]). *A* probabilistic input-preserving kernel *(or* probabilistic kernel *for short) is a function* $f \colon \mathbf{M}[X] \to \mathcal{D}\mathbf{M}[Y]$ *satisfying:*

(i) $X \subseteq Y$,

(ii) $\pi_X \circ f = \eta^{\mathcal{D}}_{\mathbf{M}[X]}$, *where* $\eta^{\mathcal{D}}_{\mathbf{M}[X]}(\mathbf{m})$ *returns the dirac distribution over* $\mathbf{m}$.

The set of all probabilistic kernels is denoted ProbKer.

The probabilistic model is a structure based on the carrier set ProbKer.

▶ **Definition 4** (Probabilistic model, [4]). *The* probabilistic frame based on Val $\mathbf{PrFr}[\mathrm{Val}]$ *(or simply* $\mathbf{PrFr}$ *when* Val *is evident) is a tuple* $\langle \mathrm{ProbKer}, \sqsubseteq, \oplus, \odot, \mathrm{ProbKer} \rangle$ *where* $\odot, \oplus, \sqsubseteq$ *are defined for arbitrary* $f \colon \mathbf{M}[X] \to \mathcal{D}\mathbf{M}[Y]$ *and* $g \colon \mathbf{M}[Z] \to \mathcal{D}\mathbf{M}[W]$ *as:*

- *the sequential composition* $f \odot g$ *is defined iff* $Y = Z$. *In this case,* $f \odot g$ *is of the form* $\mathbf{M}[X] \to \mathcal{D}\mathbf{M}[W]$, *and given* $\mathbf{m} \in \mathbf{M}[X]$, $(f \odot g)(\mathbf{m})$ *maps* $\mathbf{n} \in \mathbf{M}[W]$ *to* $\sum_{\boldsymbol{\ell} \in \mathrm{supp}(f(\mathbf{m}))} (f(\mathbf{m})(\boldsymbol{\ell}) \cdot g(\boldsymbol{\ell})(\mathbf{n}))$;
- *the parallel composition* $f \oplus g$ *is defined iff* $X \cap Z = Y \cap W$. *In this case,* $f \oplus g$ *is of the form* $\mathbf{M}[X \cup Z] \to \mathcal{D}\mathbf{M}[Y \cup W]$ *such that given* $\boldsymbol{\ell} \in \mathbf{M}[X \cup Z]$ *and* $\mathbf{m} \in \mathbf{M}[Y \cup W]$, *we have* $(f \oplus g)(\boldsymbol{\ell})(\mathbf{m}) = f(\boldsymbol{\ell}^X)(\mathbf{m}^Y) \cdot g(\boldsymbol{\ell}^Z)(\mathbf{m}^W)$;
- *the subkernel relation* $f \sqsubseteq g$ *holds if there exist a finite set of variables* S *and* $h \in \mathrm{ProbKer}$ *such that* $g = \left(f \oplus \eta^{\mathcal{D}}_{\mathbf{M}[S]}\right) \odot h$.

The probabilistic model based on Val *consists of the probabilistic frame* $\mathbf{PrFr}[\mathrm{Val}]$ *and the following* natural valuation $\mathcal{V}_{\mathrm{nat}} \colon \mathcal{AP} \to \mathcal{P}(\mathrm{ProbKer})$*: given* $(S \triangleright [T])$ *and* $f \colon \mathbf{M}[X] \to \mathcal{D}\mathbf{M}[Y]$, $f \in \mathcal{V}_{\mathrm{nat}}(S \triangleright [T])$ *iff there exists a probabilistic kernel* $f' \colon \mathbf{M}[X'] \to \mathcal{D}\mathbf{M}[Y']$ *such that* $f' \sqsubseteq f$, $X' = S$ *and* $T \subseteq Y'$.

Next we give examples of probabilistic kernels and how they compose. We write a map from a variable x to a value c as c_x and use the ket notation $a|\omega\rangle$ to denote a probabilistic outcome ω of probability a.

▶ **Example 5.** Consider variables x, y, z that take values in $\mathbf{Val} = \{0, 1\}$. We define a map $f \colon \mathbf{M}[\{z\}] \to \mathcal{D}\mathbf{M}[\{x, y, z\}]$ by:

$$f(0_z) = \frac{1}{4}|0_x, 0_y, 0_z\rangle + \frac{1}{4}|0_x, 1_y, 0_z\rangle + \frac{1}{4}|1_y, 0_y, 0_z\rangle + \frac{1}{4}|1_y, 1_y, 0_z\rangle$$
$$f(1_z) = \frac{1}{16}|0_x, 0_y, 1_z\rangle + \frac{3}{16}|0_x, 1_y, 1_z\rangle + \frac{3}{16}|1_y, 0_y, 1_z\rangle + \frac{9}{16}|1_y, 1_y, 1_z\rangle$$

Each input memory (coloured) is preserved by f so it is a probabilistic kernel. Then define $g_1 \colon \mathbf{M}[\{z\}] \to \mathcal{D}\mathbf{M}[\{x, z\}]$ and $g_2 \colon \mathbf{M}[\{z\}] \to \mathcal{D}\mathbf{M}[\{y, z\}]$ as:

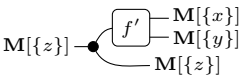
$$g_1(0_z) = \frac{1}{2}|0_x, 0_z\rangle + \frac{1}{2}|1_y, 0_z\rangle \qquad g_1(1_z) = \frac{1}{4}|0_x, 1_z\rangle + \frac{3}{4}|1_y, 1_z\rangle$$
$$g_2(0_z) = \frac{1}{2}|0_y, 0_z\rangle + \frac{1}{2}|1_y, 0_z\rangle \qquad g_2(1_z) = \frac{1}{4}|0_y, 1_z\rangle + \frac{3}{4}|1_y, 1_z\rangle$$

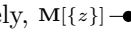
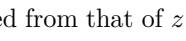
Both g_1 and g_2 are probabilistic kernels as well. The parallel composition $g_1 \oplus g_2$ is defined since $\{z\} \cap \{z\} = \{x, z\} \cap \{y, z\}$; in fact, it is easy to verify that $g_1 \oplus g_2 = f$. Moreover, g_1 and g_2 can be obtained by projecting the output of f on $\{x, z\}$ and $\{y, z\}$, respectively, and we can show $g_1 \sqsubseteq f$ and $g_2 \sqsubseteq f$.

4 DIBI models in Markov categories

In this section we construct more abstract DIBI models based on categorical structures. The starting point of our approach is a categorical characterisation of the concrete probabilistic models given above. In the following, we begin by showing examples of how elements in that model can be reformulated in categorical terms and then formally present our categorical construction of DIBI models.

As we noted in Section 1, the probabilistic DIBI kernels can be identified as morphisms in the Kleisli category for the distribution monad $\mathcal{K}\ell(\mathcal{D})$ (Definition 36); however, not all morphisms in $\mathcal{K}\ell(\mathcal{D})$ are probabilistic DIBI kernels, so we need to define the extra conditions categorically. First, we identify the $\mathcal{K}\ell(\mathcal{D})$ morphisms operating on memories. Let $\mathbb{M}\mathrm{em}\mathbb{P}\mathrm{r}$ be the subcategory of $\mathcal{K}\ell(\mathcal{D})$ where objects are restricted to memory spaces over Val. That is, the objects are memory spaces $\mathbf{m}\colon X \to \mathrm{Val}$, and the morphisms are maps $f : \mathbf{M}[X] \to \mathcal{D}\mathbf{M}[Y]$ (or $f\colon \mathbf{M}[X] \nrightarrow \mathbf{M}[Y]$ using the Kleisli category notation). Then, probabilistic kernels are exactly those morphisms in the $\mathbb{M}\mathrm{em}\mathbb{P}\mathrm{r}$ that satisfy the input-preserving condition in Definition 3. So next, we need to express the input-preserving condition categorically. To do that, we depict $\mathbb{M}\mathrm{em}\mathbb{P}\mathrm{r}$ morphisms using string diagrams, which is possible because $\mathbb{M}\mathrm{em}\mathbb{P}\mathrm{r}$ is a subcategory of the monoidal category $\mathcal{K}\ell(\mathcal{D})$ We also observe that the codomain of an input-preserving kernel $f\colon \mathbf{M}[X] \nrightarrow \mathbf{M}[Y]$ can be decomposed as $\mathbf{M}[X] \times \mathbf{M}[Y \setminus X]$. Recall the probabilistic kernel f from Example 5. Since its codomain $\mathbf{M}[\{x, y, z\}]$ can be decomposed as $\mathbf{M}[\{x\}] \times \mathbf{M}[\{y\}] \times \mathbf{M}[\{z\}]$, we can draw it as follows:



Intuitively,  produces two copies of the value of z, and the values of x and y are computed from that of z via  , while the value of z gets preserved through a straight wire in the bottom. As in this example, such copy structure of $\mathcal{K}\ell(\mathcal{D})$ enables us to capture the "input-preserving" condition of probabilistic kernels generally.

Next we want to express the sequential ($\odot$) and parallel ($\oplus$) compositions of probabilistic kernels categorically. The former is exactly the sequential composition in $\mathcal{K}\ell(\mathcal{D})$. The parallel composition, however, is *not* the monoidal product $\otimes$ in $\mathcal{K}\ell(\mathcal{D})$. By definition, the monoidal product is total, while the parallel composition is partial. Even when the parallel composition is defined, the types of the resulting morphisms do not match. Suppose that the parallel composition of $f\colon \mathbf{M}[X] \nrightarrow \mathbf{M}[Y]$ and $g\colon \mathbf{M}[U] \nrightarrow \mathbf{M}[V]$ is defined, we have

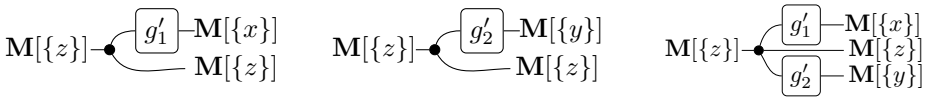
$$f \oplus g\colon \mathbf{M}[X \cup U] \nrightarrow \mathbf{M}[Y \cup V] \quad f \otimes g\colon \mathbf{M}[X] \times \mathbf{M}[U] \nrightarrow \mathbf{M}[Y] \times \mathbf{M}[V]$$

The key difference is that parallel composition considers a single memory that can be projected into two pieces, while the monoidal product considers the cartesian product of two pieces of memory, no matter if they agree or not on overlapped variables. To define the parallel composition, we need to combine $\mathbf{M}[X]$ and $\mathbf{M}[U]$ into $\mathbf{M}[X \cup U]$ categorically. Thus, we use the fact that for disjoint Z_1, Z_2, $\mathbf{M}[Z_1 \cup Z_2] \cong \mathbf{M}[Z_1] \times \mathbf{M}[Z_2]$, which implies that $\mathbf{M}[X \cup U] \cong \mathbf{M}[X \setminus U] \times \mathbf{M}[X \cap U] \times \mathbf{M}[U \setminus X]$. We illustrate the parallel composition of two probabilistic kernels from Example 5 in the following example.

▶ **Example 6.** A first way of describing parallel composition of probabilistic kernels g_1 and g_2 from Example 5 category-theoretically is by seeing them as $\mathcal{K}\ell(\mathcal{D})$-morphisms. In this setting, we may define $g_1 \oplus g_2$ as the composite

$$\begin{array}{ccc} \mathbf{M}[\{z\}] & \xrightarrow{\langle\langle \eta, g_1' \rangle, g_2' \rangle} & (\mathcal{D}\mathbf{M}[\{z\}] \times \mathcal{D}\mathbf{M}[\{x\}]) \times \mathcal{D}\mathbf{M}[\{y\}] \\ & & \downarrow \mathsf{dst} \circ \langle \mathsf{dst}, id \rangle \\ & & \mathcal{D}((\mathbf{M}[\{z\}] \times \mathbf{M}[\{x\}]) \times \mathbf{M}[\{y\}]) \xrightarrow{\mathcal{D}\cong} \mathcal{D}\mathbf{M}[\{x,y,z\}] \end{array} \tag{1}$$

where dst is the double strength of the monad $\mathcal{D}$, and $g_1' \colon \mathbf{M}[\{z\}] \nrightarrow \mathbf{M}[\{x\}]$, $g_2' \colon \mathbf{M}[\{z\}] \nrightarrow \mathbf{M}[\{y\}]$ represent the conditional distributions obtained by suitable projections of g_1 and g_2 respectively. Now consider an alternative presentation: we draw kernels g_1 and g_2 respectively as the first and second string diagrams below. The parallel composition $g_1 \oplus g_2$ is then given by the rightmost string diagram below.



(2)

The formulation (2), which we adopt in our work, has two advantages over (1). First, string diagrams make for a cleaner presentation, abstracting away most "bureaucratic" steps in (1). Second, for kernels of larger sizes, the use of diagrams drastically simplifies calcu lations, see, e.g., the verification of frame conditions in proving Theorem 12 below. e Therefore, we will define categorical DIBI models and their compositions using string diagrams, though (1) exists as an alternative formulation.

We give the formal string diagrammatic definitions of the compositions later in Definition 10, as part of the generic construction of DIBI models.

While we simply use the concept of memory spaces $\mathbf{M}[X]$ to define the subcategory $\mathbb{MemPr}$, that concept of memory spaces is customised for reasoning about probabilistic programs and relational databases and has potential to be parameterised. We observe that the side conditions of the parallel and sequential compositions are all based on comparing the set of variables in the (co)domains, so they only depend on the variable part (i.e., X) in $\mathbf{M}[X]$. This motivates us to define DIBI states as morphisms in a category whose objects are made of variables (see Definition 7) and abstracts the map between variables and corresponding memory spaces through an assignment $\theta \colon \mathrm{Var} \to \mathbf{ob}(\mathbb{C})$, for some Markov category $\langle \mathbb{C}, \otimes, \mathsf{I} \rangle$.

Finally, we need to express finite *sets* of variables and the union of *disjoint* such sets in a monoidal category, where the monoidal products of objects do not take care of deduplication. To address that, we impose a linear order $\preceq$ on Var such that indexed variables inherit the order of their indices, e.g., $x_1 \preceq x_2 \preceq x_3$. Let $x \prec y$ abbreviate for $x \preceq y$ and $x \neq y$. Then, finite sets of variables can be represented as finite lists of variables ordered by $\prec$, via a translation that we write as $[\![\cdot]\!]$. For instance, $[\![\{x_3, x_1, x_3, x_4\}]\!] = [x_1, x_3, x_4]$. This will be realised in two steps: we first define a category whose objects are finite lists of variables (Definition 7), and then we restrict the objects to finite lists without duplicates that respect the linear order (Definition 9).

Now we are ready to define a symmetric monoidal category $\mathbb{C}[\theta]$ that has enough structure to support our categorical characterisation of DIBI models. The category $\mathbb{C}[\theta]$ is parameterised by $\mathbb{C}$, whose objects abstract the concept of memory spaces. For simplicity, we fix a Markov category $\mathbb{C}$ throughout the rest of the section.

▶ **Definition 7.** *Let $\mathbb{C}[\theta]$ be the symmetric monoidal category whose objects are finite lists of variables, and morphisms $[x_1, \ldots, x_m] \to [y_1, \ldots, y_n]$ are $\mathbb{C}$-morphisms $\theta(x_1) \otimes \cdots \otimes \theta(x_m) \to \theta(y_1) \otimes \cdots \otimes \theta(y_n)$. Sequential composition is defined as in $\mathbb{C}$. The identity on $[x_1, \ldots, x_m]$ is $id_{\theta x_1 \otimes \cdots \otimes \theta x_m}$. The monoidal product in $\mathbb{C}[\theta]$ – which we also write as $\otimes$ with abuse of notation – is list concatenation on objects, and monoidal product in $\mathbb{C}$ on morphisms.*

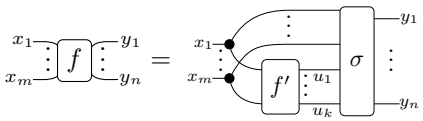
The SMC and Markov category structure of $\mathbb{C}[\theta]$ follow from those of $\mathbb{C}$. In particular, the symmetric map $[x_1,\dots,x_m]\otimes[y_1,\dots,y_n]\to[y_1,\dots,y_n]\otimes[x_1,\dots,x_m]$ is the symmetry morphism $(\theta(x_1)\otimes\cdots\otimes\theta(x_m))\otimes(\theta(y_1)\otimes\cdots\otimes\theta(y_n))\to(\theta(y_1)\otimes\cdots\otimes\theta(y_n))\otimes(\theta(x_1)\otimes\cdots\otimes\theta(x_m))$ in $\mathbb{C}$. The copy map $[x_1,\dots,x_m]\to[x_1,\dots,x_m]\otimes[x_1,\dots,x_m]=[x_1,\dots,x_m,x_1,\dots,x_m]$ is the copy map $\mathsf{copy}_{\theta(x_1)\otimes\cdots\otimes\theta(x_m)}\colon\theta(x_1)\otimes\cdots\otimes\theta(x_m)\to(\theta(x_1)\otimes\cdots\otimes\theta(x_m))\otimes(\theta(x_1)\otimes\cdots\otimes\theta(x_m))$ in $\mathbb{C}$. The tensor unit in $\mathbb{C}[\theta]$ is the empty list $[\,]$, and θ maps it to the tensor unit I of the SMC $\mathbb{C}$. The delete map $[x_1,\dots,x_m]\to[\,]$ is then the delete map $\mathsf{del}_{\theta(x_1)\otimes\cdots\otimes\theta(x_m)}\colon\theta(x_1)\otimes\cdots\otimes\theta(x_m)\to\mathsf{I}$ in $\mathbb{C}$.

Sometimes we restrict ourselves to a uniform assignment θ; that is, for some fixed $\mathsf{C}\in\mathbf{ob}(\mathbb{C})$, $\theta(x)=\mathsf{C}$ for all $x\in\mathrm{Var}$. This is in line with the scenario where a fixed value space Val is used for all variables (see Definition 3). In this case, we write $\mathbb{C}[\theta]$ as $\mathbb{C}[\mathsf{C}]$ to emphasise the uniform value of the assignment. This category can be seen as the full subcategory of $\mathbb{C}$ freely generated by C, but with each occurrence of the generating object named by a variable. The next example shows how the construction in Definition 7 selects morphisms of $\mathcal{KL}(\mathcal{D})$ that act on memory spaces, among which we have all the probabilistic kernels.

▶ **Example 8.** Let $\mathbb{C}$ be $\mathcal{KL}(\mathcal{D})$, and $\theta\colon\mathrm{Var}\to\mathbf{ob}(\mathcal{KL}(\mathcal{D}))$ be the constant function $x\mapsto\mathrm{Val}$ for all $x\in\mathrm{Var}$. Then there is a full and faithful embedding functor $\iota\colon\mathbb{MemPr}\to\mathcal{KL}(\mathcal{D})[\theta]$: on objects, given a set X, $\iota(\mathbf{M}[X])=[\![X]\!]$; on morphisms, given $f\colon\mathbf{M}[X]\to\mathcal{D}\mathbf{M}[Y]$ with $X=\{x_1,\dots,x_m\}$ and $Y=\{y_1,\dots,y_n\}$, its image $\iota(f)\colon[\![X]\!]\to[\![Y]\!]$ is the composed map $\mathrm{Val}^m\xrightarrow{\cong}\mathbf{M}[X]\xrightarrow{f}\mathcal{D}\mathbf{M}[Y]\xrightarrow{\mathcal{D}\cong}\mathcal{D}\mathrm{Val}^n$, where the isomorphisms are, e.g., $\mathbf{M}[Y]\xrightarrow{\cong}\mathbf{M}[\{y_1\}]\times\cdots\times\mathbf{M}[\{y_n\}]\xrightarrow{\cong^\theta}\mathrm{Val}^n$, using the valuation $\theta(y_j)=\mathrm{Val}$.

Just as the states of the probabilistic models are exactly input-preserving morphisms in $\mathbb{MemPr}$, we define the notion of *input-preserving kernels* in $\mathbb{C}[\theta]$, written $\mathrm{Ker}(\mathbb{C}[\theta])$ and use them as the states of our categorical DIBI models.

▶ **Definition 9.** *A $\mathbb{C}[\theta]$-morphism $f\colon[x_1,\dots,x_m]\to[y_1,\dots,y_n]$ is a* $\mathbb{C}[\theta]$ input-preserving kernel *(or* $\mathbb{C}[\theta]$-kernel *for short) if $x_1\prec\cdots\prec x_m$, $y_1\prec\cdots\prec y_n$, and f can be decomposed as follows, where σ is a rewiring:*

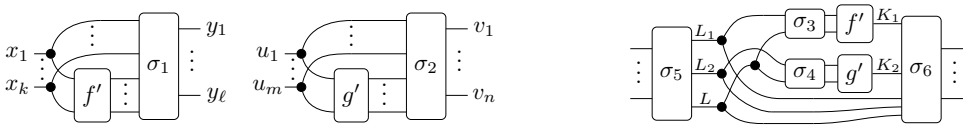


(3)

In words, a $\mathbb{C}[\theta]$-kernel is a morphism whose interfaces are essentially finite sets of variables, such that the input is preserved as part of the output (through the upper leg of those ⊸s). The map f' in (3) is referred to as the nontrivial part of the input-preserving kernel. It follows from Definition 9 that, for a $\mathbb{C}[\theta]$-kernel, its codomain $[y_1,\dots,y_n]$ always subsumes its domain $[x_1,\dots,x_m]$; also, $u_1,\dots,u_k$ are precisely those y_js that are not among these x_is. Since the (co)domains of $\mathbb{C}[\theta]$-kernel are list presentation of sets, we also write the types of $\mathbb{C}[\theta]$-kernels using the corresponding sets, e.g., in (3), $f\colon\{x_1,\dots,x_m\}\to\{y_1,\dots,y_n\}$.

Next we define compositions on input-preserving kernels, generalising what we have seen in Example 6 for the probabilistic models.

▶ **Definition 10** (Compositions). *Given arbitrary $\mathbb{C}[\theta]$-kernels $f\colon X\to Y$ and $g\colon U\to V$ as in Figure 3a, their* sequential composition *$f\odot g$ is defined iff $cod(f)=dom(g)$, in which case $f\odot g=g\circ f$. Their* parallel composition *$f\oplus g$ is defined iff $X\cap U=Y\cap V$. Assume $L=[\![X\cap U]\!]$, $L_1=[\![X\setminus(X\cap U)]\!]$, $L_2=[\![U\setminus(X\cap U)]\!]$, $K_1=[\![Y\setminus(Y\cap V)]\!]$, and $K_2=[\![V\setminus(Y\cap V)]\!]$, then $f\oplus g\colon X\cup U\to Y\cup V$ is defined as in Figure 3b, where all the σ_is are rewiring morphisms for making the input and output variables $\prec$-ordered.*

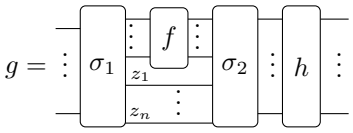


(a) $\mathbb{C}[\theta]$-kernels f and g. **(b)** $f \oplus g$.

Figure 3 Parallel composition of $\mathbb{C}[\theta]$.

Note here a benefit of the diagrammatic representation: we can easily identify the memory overlap $\mathbf{M}[X \cap Y]$, as it is depicted a separate wire; with traditional syntax, we would need to apply associativity and commutativity to extract it from $\mathbf{M}[X \cup Y]$. It is easy to see that kernels are closed under compositions. Also, for curious readers, $\mathbb{C}[\theta]$-kernels with their parallel compositions form a *partially monoidal category* [2]. Next we define the subkernel relation.

▶ **Definition 11** (Subkernel). *Given two $\mathbb{C}[\theta]$-kernels f and g, we say f is a* subkernel *of g – denoted as $f \sqsubseteq g$ – if there exist $z_1, \ldots, z_n \in \mathrm{Var}$, a $\mathbb{C}[\theta]$-kernel h, and rewiring morphisms σ_1, σ_2 such that g can be expressed as follows:*



The subkernel relation is transitive and reflexive, which can be shown simply by manipulations of the string diagram. We are finally able to state the main result of this section: $\mathbb{C}[\theta]$-kernels and their compositions form a DIBI frame.

▶ **Theorem 12.** $\mathbf{Fr}(\mathbb{C}[\theta]) = \langle \mathrm{Ker}(\mathbb{C}[\theta]), \sqsubseteq, \oplus, \odot, \mathrm{Ker}(\mathbb{C}[\theta]) \rangle$ *is a DIBI frame.*

Also, under the natural valuation $\mathcal{V}_{\mathrm{nat}}$, a $\mathbb{C}[\theta]$-kernel $f\colon X \to Y$ satisfies $S \triangleright [T]$ iff there is a subkernel $(f'\colon X' \to Y') \sqsubseteq f$ such that $X' = S$ and $Y' \supseteq T$. Thus:

▶ **Corollary 13.** $(\mathbf{Fr}(\mathbb{C}[\theta]), \mathcal{V}_{\mathrm{nat}})$ *is a DIBI model.*

We will see in Section 5 how to use this categorical construction to derive a wide range of DIBI models. Moreover, it also enables us to extract the conditions needed for a specific feature of a DIBI model as properties of the underlying category. Here is an example.

▶ **Proposition 14.** *If $\mathbb{C}$ further satisfies that for arbitrary morphisms f, g and object D, $f \otimes \mathsf{del}_{\mathsf{D}} = g \otimes \mathsf{del}_{\mathsf{D}}$ implies $f = g$, then a subkernel is unique given its type in the following sense: if $\mathbb{C}[\theta]$-kernels $f_1, f_2\colon U \to V$ are both subkernels of g, then $f_1 = f_2$.*

Note that the uniqueness of subkernels has been observed already in the context of probabilistic and relational models, see [4, Sect. IV]. Proposition 14 reveals the general conditions under which this uniqueness holds for a wider class of DIBI models.

5 Examples

In this section we provide concrete instances of the categorical construction in Section 4. The first example recovers the probabilistic DIBI models. The remaining examples are new DIBI models. Some of them have been suggested in the DIBI paper [4], yet not materialised due to the complexity involved in stating each component and verifying the frame conditions. Within our framework, these steps become much easier to perform.

5.1 Probabilistic (and Relational) DIBI Models

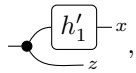
As we sketched in Example 6 and Example 8, the probabilistic DIBI kernels and $\langle \mathbf{Fr}(\mathcal{K}\ell(\mathcal{D})), \mathcal{V}_{\text{nat}} \rangle$ input-preserving kernels correspond to each other. We now formally show that the probabilistic DIBI model in Definition 4 can be recovered from the categorical DIBI model $\langle \mathbf{Fr}(\mathcal{K}\ell(\mathcal{D})), \mathcal{V}_{\text{nat}} \rangle$. Since both models are equipped with the natural valuation $\mathcal{V}_{\text{nat}}$, we focus on the frame part. To make the correspondence precise, we introduce the category of DIBI frames, as hinted in [4, Sect. III].

▶ **Definition 15.** *In* the category of DIBI frames $\mathbb{DibiFr}$*, objects are DIBI frames; morphisms* $f : \langle S, \sqsubseteq_S, \oplus_S, \odot_S, E_S \rangle \to \langle T, \sqsubseteq_T, \oplus_T, \odot_T, E_T \rangle$ *are functions* $f : S \to T$ *that respect all the relations and partial operations: for arbitrary* $s, s' \in S$*,*

- $s \sqsubseteq_S s'$ *implies* $f(s) \sqsubseteq_T f(s')$*;*
- *if* $s \star_S s'$ *is defined, then* $f(s) \star_S f(s')$ *is defined, and* $f(s) \star_T f(s') = f(s \star_S s')$*, for* $\star \in \{\oplus, \odot\}$*;*
- $s \in E_S$ *implies* $f(s) \in E_T$*.*

It turns out that the function ι introduced in Example 8 extends to an isomorphism of DIBI frames from $\mathbf{PrFr}[\text{Val}]$ to $\mathbf{Fr}(\mathcal{K}\ell(\mathcal{D})[\text{Val}])$.

▶ **Proposition 16.** $\mathbf{PrFr}[\text{Val}] \cong \mathbf{Fr}(\mathcal{K}\ell(\mathcal{D})[\text{Val}])$.

▶ **Example 17.** The probabilistic kernel $g_1 : \mathbf{M}[\{z\}] \to \mathcal{D}\mathbf{M}[\{x, z\}]$ from Example 5 corresponds to the following $\mathcal{K}\ell(\mathcal{D})[\{0,1\}]$-kernel $h_1 : [z] \to [x, z]$ – i.e., a $\mathcal{K}\ell(\mathcal{D})$-morphism $\{0,1\} \nrightarrow \{0,1\}^2$ – where: $0 \mapsto \frac{1}{2}|0,0\rangle + \frac{1}{2}|1,0\rangle$, $1 \mapsto \frac{1}{4}|0,1\rangle + \frac{3}{4}|1,1\rangle$. Diagrammatically, h_1 is of the form 

, where $h_1' : [z] \to [x]$ is the map such that $0 \mapsto \frac{1}{2}|0\rangle + \frac{1}{2}|1\rangle$ and $1 \mapsto \frac{1}{4}|0\rangle + \frac{3}{4}|1\rangle$

Similarly, the relational DIBI model from [4] with the value space Val can be shown to be isomorphic to $\mathbf{Fr}(\mathcal{K}\ell(\mathcal{P}_i)[\text{Val}])$, where $\mathcal{P}_i$ is the nonempty powerset monad A.

5.2 Stochastic DIBI Models

Using our categorical construction, we can derive a notion of DIBI model for continuous probabilistic (stochastic) processes, not previously considered. This is of interest because, as we show in Section 6, it allows to capture conditional independence for continuous probability using DIBI formulas. We take as underlying category $\mathbb{Stoch}$ of stochastic processes, defined as the Kleisli category $\mathcal{K}\ell(\mathcal{G})$ for the Giry monad on measurable spaces – see Appendix A for a full definition. Since $\mathcal{G}$ is an affine symmetric monoidal monad, $\mathbb{Stoch}$ is a Markov category. Applying Theorem 12 to $\mathbb{C} = \mathbb{Stoch}$, we get DIBI frames based on stochastic processes.

▶ **Proposition 18.** *Given an arbitrary map* $\theta : \text{Var} \to \mathbf{ob}(\mathbb{Meas})$*,* $\mathbf{Fr}(\mathbb{Stoch}[\theta]) = \langle \text{Ker}(\mathbb{Stoch}[\theta]), \sqsubseteq, \oplus, \odot, \text{Ker}(\mathbb{Stoch}[\theta]) \rangle$ *is a DIBI frame.*

We call $\mathbf{Fr}(\mathbb{Stoch}[\theta])$ the *stochastic DIBI frame* based on θ and elements in $\text{Ker}(\mathbb{Stoch}[\theta])$ stochastic kernels.

▶ **Example 19.** We show a representation of the *Box-Muller transformation* using stochastic kernels. Consider θ that maps all variable names to the Borel σ-algebra over reals $(\mathbb{R}, \mathcal{B}(\mathbb{R}))$. Define stochastic kernels $g_1 : \varnothing \to \{u\}$ and $g_2 : \varnothing \to \{w\}$ – both standing for $\mathbb{Stoch}$-morphisms $(\mathbf{1}, \{\varnothing, \mathbf{1}\}) \to (\mathbb{R}, \mathcal{B}(\mathbb{R}))$, or equivalently, a probabilistic measure on $(\mathbb{R}, \mathcal{B}(\mathbb{R}))$ – by $g_i(\bullet) = \text{Unif}(0,1)$ for $i = 1, 2$, where $\text{Unif}(0,1)$ is the uniform measure over the interval $(0,1)$. Such

a uniform measure over infinite outcomes is not possible in the discrete probabilistic DIBI model. Define another stochastic kernel $f : \{u, w\} \to \{u, w, x, y\}$ where the value of x, y are determined by the value of u, w:

$$f(u \mapsto v_u, w \mapsto v_w) = \delta_{v_u, v_w, \left(\sqrt{-2\ln u}\cdot\cos(2\pi w)\right)_x, \left(\sqrt{-2\ln u}\cdot\sin(2\pi w)\right)_y}.$$

Then $h = (g_1 \oplus g_2) \odot f$ gives a stochastic kernel $\varnothing \to \{u, w, x, y\}$. Box-Muller transformation says that x and y are independent in $h(\langle\rangle)$ despite their seemingly dependence on u and w.

Comparison with Lilac [24]. Our stochastic DIBI models can be used to reason about independence and conditional probabilities in continuous distributions. A recent work Lilac by Li et al. [24] proposed a BI model for the same goal, yet with some crucial differences in the set-up.

First, the states in Lilac's BI model are probabilistic space fragments of a fixed sample space, and their variables are mathematical random variables that deterministically map elements in the sample space to values. In comparison, we treat variables as names that can be associated to values or distributions. Our stochastic kernels – though not using an ambient sample space – can encode their set-up: we can devise a special variable Ω for "the sample space", and deterministic kernels from Ω to other variables encode random variables.

Second, to reason about conditional probabilities, Lilac want probability spaces to be disintegrable with respect to well-behaved random variables. To achieve that, they require probability spaces in their model to be extensible to Borel spaces, since disintegration works nicer in Borel spaces. By working with kernels, which already represent conditional probability spaces, we do not need to impose disintegratability on our DIBI states to reason about conditional probabilities. For instance, while disintegration is not always possible in the category $\mathbb{S}\mathtt{toch}$, we can still construct a DIBI model based on $\mathbb{S}\mathtt{toch}$.

Other measure-theoretic probabilistic DIBI models. The category $\mathbb{S}\mathtt{toch}$ is not the only Markov category for measure-theoretic probability. Another choice is $\mathbb{B}\mathtt{orel}\mathbb{S}\mathtt{toch}$, a subcategory of $\mathbb{S}\mathtt{toch}$ obtained by restricting to standard Borel spaces as objects. It has some nice properties that $\mathbb{S}\mathtt{toch}$ does not satisfy, such as having conditionals as mentioned above. $\mathbb{B}\mathtt{orel}\mathbb{S}\mathtt{toch}$ is also a Markov category and we can easily instantiate a DIBI model.

▶ **Proposition 20.** *Given any map* $\theta\colon \mathrm{Var} \to \mathbf{ob}(\mathbb{B}\mathtt{orel}\mathbb{S}\mathtt{toch})$, $\mathbf{Fr}(\mathbb{B}\mathtt{orel}\mathbb{S}\mathtt{toch}[\theta])$ *defined as* $\langle \mathrm{Ker}(\mathbb{B}\mathtt{orel}\mathbb{S}\mathtt{toch}[\theta]), \sqsubseteq, \oplus, \odot, \mathrm{Ker}(\mathbb{B}\mathtt{orel}\mathbb{S}\mathtt{toch}[\theta])\rangle$ *is a DIBI frame.*

The study of measure theory is also intertwined with topology, and another category for measure-theoretic probability is the Kleisli category of the *Radon monad* $\mathcal{R}$ based on the category of compact Hausdorff spaces $\mathbb{CH}\mathtt{aus}$ and continuous maps, which we denote as $\mathcal{K}\ell_{\mathbb{CH}\mathtt{aus}}(\mathcal{R})$. $\mathcal{K}\ell_{\mathbb{CH}\mathtt{aus}}(\mathcal{R})$ is also a Markov category [12], so Theorem 12 applies.

▶ **Proposition 21.** *Given any map* $\theta : \mathrm{Var} \to \mathbf{ob}(\mathcal{K}\ell_{\mathbb{CH}\mathtt{aus}}(\mathcal{R}))$, $\mathbf{Fr}(\mathcal{K}\ell_{\mathbb{CH}\mathtt{aus}}(\mathcal{R})[\theta])$ *defined as* $\langle \mathrm{Ker}(\mathcal{K}\ell_{\mathbb{CH}\mathtt{aus}}(\mathcal{R})[\theta]), \sqsubseteq, \oplus, \odot, \mathrm{Ker}(\mathcal{K}\ell_{\mathbb{CH}\mathtt{aus}}(\mathcal{R})[\theta])\rangle$ *is a DIBI frame.*

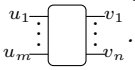
A measure-theoretic Markov category not formed as Kleisli categories is the Gaussian probability category $\mathbb{G}\mathtt{auss}$ [12]. Its objects are natural numbers, and a morphism $n \to m$ is a tuple (M, σ^2, μ) representing the function $f : \mathbb{R}^n \to \mathbb{R}^m$ with $f(v) = M \cdot v + \xi$, where ξ is the Gaussian noise with mean μ and covariance matrix σ^2. Its monoidal product is addition $+$ on the objects and vector concatenation on morphisms. $\mathbb{G}\mathtt{auss}$ differs from $\mathbb{S}\mathtt{toch}$, $\mathbb{B}\mathtt{orel}\mathbb{S}\mathtt{toch}$ and $\mathcal{K}\ell_{\mathbb{CH}\mathtt{aus}}(\mathcal{R})$ in that it does not arise as the Kleisli category associated to some monad. But since it is a Markov category, we can again instantiate DIBI models based on $\mathbb{G}\mathtt{auss}$.

▶ **Proposition 22.** *Given any map* θ : Var $\to$ **ob**($\mathbb{G}\mathtt{auss}$), **Fr**($\mathbb{G}\mathtt{auss}[\theta]$) *defined as* $\langle \mathrm{Ker}(\mathbb{G}\mathtt{auss}[\theta]), \sqsubseteq, \oplus, \odot, \mathrm{Ker}(\mathbb{G}\mathtt{auss}[\theta])\rangle$ *is a DIBI frame.*

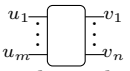
5.3 Syntactic DIBI Models

The DIBI models defined so far all have kernels defined by some processes over memory spaces. It is worth considering a different flavour: purely formal, syntactically generated DIBI models. We start by defining the underlying category.

▶ **Definition 23.** $\mathbb{S}\mathtt{yn}\mathbb{V}\mathtt{ar}$ *is the Markov category freely generated as follows:*

- *the generating objects are variables in* Var*;*
- *there is exactly one generating morphism of type* $[u_1, \dots, u_m] \to [v_1, \dots, v_n]$ *for distinct variables* $u_1 \prec \cdots \prec u_m$ *and* $v_1 \prec \cdots \prec v_n$*, written as string diagrams of the form* 

.

In words, $\mathbb{S}\mathtt{yn}\mathbb{V}\mathtt{ar}$-objects are finite lists of variables (without the requirements of duplicate-free or $\preceq$-ordered); morphisms are diagrams freely concatenated using —,  ,  ,  , and 

(quotiented by the Markov category equations). The syntactic DIBI frame is based on the category $\mathbb{S}\mathtt{yn}\mathbb{V}\mathtt{ar}[id]$, where id: Var $\to$ **ob**($\mathbb{S}\mathtt{yn}\mathbb{V}\mathtt{ar}$) maps $x \in$ Var to the singleton list $[x]$.

▶ **Proposition 24.** **SynFr** $= \langle \mathrm{Ker}(\mathbb{S}\mathtt{yn}\mathbb{V}\mathtt{ar}[id]), \sqsubseteq, \oplus, \odot, \mathrm{Ker}(\mathbb{S}\mathtt{yn}\mathbb{V}\mathtt{ar}[id])\rangle$ *is a DIBI frame.*

Equipped with the natural valuation $\mathcal{V}_{\mathrm{nat}}$, one obtains a DIBI model $\langle \mathbf{SynFr}, \mathcal{V}_{\mathrm{nat}}\rangle$. We postpone an example of $\mathbb{S}\mathtt{yn}\mathbb{V}\mathtt{ar}[id]$-kernels till Section 6, Example 33, in which $\mathbb{S}\mathtt{yn}\mathbb{V}\mathtt{ar}[id]$-kernels are used to distinguish two notions of conditional independence in Markov categories.

An interesting question for future work is how to extend the syntactic DIBI model to a term model. Typically being initial objects in categories of models, term models can help proving completeness and defining categorical semantics for formal systems, including algebraic theories [23], logics [38] (e.g., Lindenbaum–Tarski algebras) and type theories [19, 18]. A term model for DIBI could lead to a sound and complete axiomatisation of the specific version of DIBI logic in this paper, whose atomic propositions take the form of $S \triangleright [T]$.

6 Conditional independence

DIBI logic is designed for reasoning about conditional independence (CI). The prior work [4] shows that, CI in the discrete probabilistic models and join dependency in the relational models can be characterised by the same class of DIBI formulas. Generalising this result, in this section we define a notion of CI on $\mathbb{C}[\theta]$-kernels based on formula satisfaction. Since $\mathbb{C}[\theta]$ is a Markov category, we can compare our logical notion of CI with existing categorical definitions of CI in Markov categories [9, 12].

Fix a Markov category $\mathbb{C}$ and a map θ: Var $\to$ **ob**($\mathbb{C}$). We define CI in the DIBI model $\langle \mathbf{Fr}(\mathbb{C}[\theta]), \mathcal{V}_{\mathrm{nat}}\rangle$.

▶ **Definition 25** (Conditional Independence)**.** *For any mutually disjoint finite sets of variables* W, X, Y, U*,* X *and* Y *are* DIBI conditionally independent given W *in a* $\mathbb{C}[\theta]$*-kernel*[3] $f : \varnothing \to W \cup X \cup Y \cup U$ *(denoted as* $X \perp\!\!\!\perp_{\mathrm{L}} Y \,|\, W$*) if*

$$f \vDash_{\mathcal{V}_{\mathrm{nat}}} (\varnothing \triangleright [W]) \fatsemi ((W \triangleright [X]) * (W \triangleright [Y])). \tag{4}$$

[3] Note that $\mathbb{C}[\theta]$-kernels with domain $\varnothing$ are not to be thought of as maps with empty domains. For instance, $\mathcal{K}\ell(\mathcal{D})[\theta]$-kernels of the form $\varnothing \to \{x, y\}$ corresponds to $\mathcal{K}\ell(\mathcal{D})$-morphisms $\mathbf{1} \nrightarrow \theta(x) \times \theta(y)$, which denote distributions over x, y.

Let us unfold what (4) means. Under the natural valuation $\mathcal{V}_{\text{nat}}$, the atomic proposition $S \triangleright [T]$ encodes the dependence of T on S: formally, a $\mathbb{C}[\theta]$-kernel $f\colon X \to Y$ satisfies $S \triangleright [T]$ iff f contains some subkernel $f'\colon S \to Y'$ such that $T \subseteq Y'$. So the formula in (4) requires that the kernel f has empty domain and can be decomposed as $f \sqsupseteq f_0 \odot (f_1 \oplus f_2)$, where f_0 determines the value on W, f_1 and f_2 determine the value on X and Y given the value on W, respectively, and f_1 and f_2 do so independently of each other. We illustrate the formula with examples in the discrete probabilistic DIBI model and the stochastic DIBI model.

▶ **Example 26.** In the setting of Example 5, consider the probabilistic kernel $h\colon \mathbf{M}[\varnothing] \to \mathcal{D}\mathbf{M}[\{x, y, z\}]$ such that :

$$h(\emptyset) = \frac{1}{8}|0_x, 0_y, 0_z\rangle + \frac{1}{8}|0_x, 1_y, 0_z\rangle + \frac{1}{8}|1_y, 0_y, 0_z\rangle + \frac{1}{8}|1_y, 1_y, 0_z\rangle + \frac{1}{32}|0_x, 0_y, 1_z\rangle + \frac{3}{32}|0_x, 1_y, 1_z\rangle + \frac{3}{32}|1_y, 0_y, 1_z\rangle + \frac{9}{32}|1_y, 1_y, 1_z\rangle$$

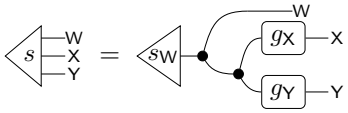
Then $h \vDash_{\mathcal{V}_{\text{nat}}} (\varnothing \triangleright [\{z\}]) ⨟ ((\{z\} \triangleright [\{z, x\}]) * (\{z\} \triangleright [\{z, y\}]))$, because $h = h_0 \odot f = h_0 \odot (g_1 \oplus g_2)$, where h_0 denotes the uniform distribution $\frac{1}{2}|0_z\rangle + \frac{1}{2}|1_z\rangle$.

▶ **Example 27.** Define g_1, g_2, f, h as in Example 19. We want to assert that variables x and y are independent in the distribution constructed by Box-Muller Transform. Independence is a special case of conditional independence in which the set of conditioned variables is empty. Thus, the goal is to assert $(\varnothing \triangleright [\varnothing]) ⨟ ((\varnothing \triangleright [\{x\}]) * (\varnothing \triangleright [\{y\}]))$ – equivalently, $(\varnothing \triangleright [\{x\}]) * (\varnothing \triangleright [\{y\}])$.

Define $h_1 : \varnothing \to \{x\}$ and $h_2 : \varnothing \to \{y\}$ both as the standard normal distribution $\mathcal{N}(0, 1)$. Clearly $h_1 \vDash_{\mathcal{V}_{\text{nat}}} \varnothing \triangleright [\{x\}]$ and $h_2 \vDash_{\mathcal{V}_{\text{nat}}} \varnothing \triangleright [\{y\}]$. Moreover, some non-trivial calculations show that $(h_1 \oplus h_2) \sqsubseteq h$, and consequently $h \vDash_{\mathcal{V}_{\text{nat}}} (\varnothing \triangleright [\{x\}]) * (\varnothing \triangleright [\{y\}])$ by definition.

Since the categorical DIBI models are based on Markov categories, we compare our logical notion of CI on kernels with the canonical notion of CI in Markov categories, which defines CI as decomposability of morphisms. Fix a Markov category $\mathbb{X}$ in Definitions 28, 31, and 34.

▶ **Definition 28.** *An $\mathbb{X}$-morphism $s\colon \mathsf{I} \to \mathsf{W} \otimes \mathsf{X} \otimes \mathsf{Y}$ displays the conditional independence of X and Y given W if there exist $\mathbb{X}$-morphisms $s_{\mathsf{W}}\colon \mathsf{I} \to \mathsf{W}$, $g_{\mathsf{X}}\colon \mathsf{W} \to \mathsf{X}$, $g_{\mathsf{Y}}\colon \mathsf{W} \to \mathsf{Y}$ such that the following equation holds. We write this as $\mathsf{X} \perp \mathsf{Y} \,|\, \mathsf{W}$.*



In the context of DIBI models, Definition 28 restricts to stating the conditional independence of X and Y given W in $\mathbb{C}[\theta]$-kernels of the form $\varnothing \to W \cup X \cup Y$. In particular, no extra variable (as that U in Definition 25) in the kernel's codomain is allowed.

▶ **Example 29.** We show an example of this notion of CI in the Markov category $\mathbb{G}\text{auss}$. Consider a morphism $s : \varnothing \to \{w, x, y\}$ specified by the tuple $\left(!, \sigma^2 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix}, \mu = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}\right)$, where ! denotes the trivial map from empty domain. That is, s takes a length 0 vector and generates a length 3 vector, holding the values of w, x and y, with the normal distribution $\mathcal{N}(\mu, \sigma^2)$. This s can be decomposed as in Definition 28 with $s_w = (!, 0, 1)$, $g_x = (1, 0, 1)$, and $g_y = (1, 0, 1)$: composing s_w, g_x and g_y, we get $\mathbb{E}(w) = \mathbb{E}(\xi_w) = 0$, $\mathbb{E}(x) = \mathbb{E}(w + \xi_x) = 0 + 0 = 0$, and $\mathbb{E}(y) = \mathbb{E}(w + \xi_y) = 0$, justifying the noise's mean μ being a zero vector. For the covariance matrix, let $v = (w, x, y) - (\mathbb{E}(w), \mathbb{E}(x), \mathbb{E}(y))$. Then $\sigma^2 = \mathbb{E}(v \cdot v^T) = \mathbb{E}((w, x, y) \cdot (w, x, y)^T)$, and one may show that σ^2 is equal to the matrix above.

(a) Markov CI. **(b)** superset CI.

Figure 4 Two possible extension of plain CI.

▶ **Proposition 30.** *For any $\mathbb{C}[\theta]$-kernel $s\colon \varnothing \to W \cup X \cup Y$ where W, X, Y are mutually disjoint, $X \perp Y \,|\, W$ iff $X \perp\!\!\!\perp_{\mathrm{L}} Y \,|\, W$.*

In order to extend Proposition 30 to the scenario in Definition 25 where a kernel f might contain some U that does not appear in the CI statement in its codomain, we need to modify the notion of CI from Definition 28 – referred to as plain CI – to allow objects that do not appear in the CI statement to occur in the codomain of s. We suggest two sensible extensions.

▶ **Definition 31.** *Given an $\mathbb{X}$-morphism $s\colon \mathsf{I} \to \mathsf{W} \otimes \mathsf{X} \otimes \mathsf{Y} \otimes \mathsf{U}$,*

- *s displays* Markov CI*, denoted $\mathsf{X} \perp_{\mathrm{M}} \mathsf{Y} \,|\, \mathsf{W}$, if there exist $s_{\mathsf{W}}, g_{\mathsf{X}}, g_{\mathsf{Y}}$ satisfying 4a.*
- *s displays* superset CI*, denoted $\mathsf{X} \perp_{\mathrm{S}} \mathsf{Y} \,|\, \mathsf{W}$, if there exist s_0, g_1, g_2 satisfying 4b.*

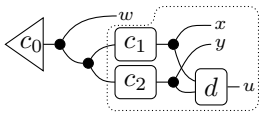
These two notions differ regarding to the treatment of the extra object U. In Figure 4a, we project out the extra object U and reduce the situation to that of Definition 28. In Figure 4b, U is kept and passed along through s_0, g_1, g_2. Clearly, both reduce to Definition 28 when no such U appears. We can now state that DIBI CI coincides with Markov CI, but is weaker than superset CI.

▶ **Theorem 32.** *Given the $\mathbb{C}[\theta]$-kernel $f\colon \varnothing \to W \cup X \cup Y \cup U$ from Definition 25,*

1. *f satisfies $X \perp_{\mathrm{M}} Y \,|\, W$ if and only if it satisfies $X \perp\!\!\!\perp_{\mathrm{L}} Y \,|\, W$;*
2. *if f satisfies $X \perp_{\mathrm{S}} Y \,|\, Z$, then it also satisfies $X \perp\!\!\!\perp_{\mathrm{L}} Y \,|\, Z$.*

Item 1 follows straightforwardly by unpacking the definitions. Item 2 follows from Item 1 and that $\mathsf{X} \perp_{\mathrm{S}} \mathsf{Y} \,|\, \mathsf{W}$ implies $\mathsf{X} \perp_{\mathrm{M}} \mathsf{Y} \,|\, \mathsf{W}$: simply apply $-\!\bullet_{\mathsf{U}}$ on both sides of Figure 4b, and Figure 4a follows via naturality of $-\!\bullet$. The converse of Item 2 does not hold in general, as demonstrated below.

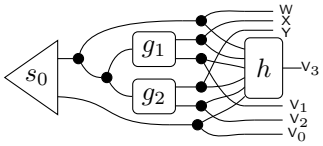
▶ **Example 33.** Consider the syntactic DIBI model $\langle \mathbf{SynFr}, \mathcal{V}_{\mathrm{nat}} \rangle$ from Section 5.3. Define the $\mathbb{S}\mathtt{yn}\mathbb{V}\mathtt{ar}[id]$-kernel f as follows, where c_0, c_1, c_2, d are all generating morphisms, i.e., not further decomposable:



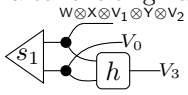
Then f satisfies the DIBI CI $x \perp\!\!\!\perp_{\mathrm{L}} y \,|\, w$, but not the superset CI $x \perp_{\mathrm{S}} y \,|\, w$: one cannot rewrite the diagram in the dotted box into a juxtaposition of two diagrams with output wires containing x and y, respectively; in other words, it cannot be rewritten as the style in Figure 4b.

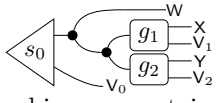
Example 33 gives some hint at how to weaken the superset CI to match DIBI CI: one needs to allow some morphism d following the morphism witnessesing $x \perp_{\mathrm{S}} y \,|\, z$. We formalise this idea and show the resulting notion is indeed equivalent to both Markov and DIBI CI.

▶ **Definition 34.** *An $\mathbb{X}$-morphism $s\colon \mathsf{I} \to \mathsf{W} \otimes \mathsf{X} \otimes \mathsf{Y} \otimes \mathsf{U}$* displays the extended superset conditional independence – *denoted as $\mathsf{X} \perp_{\mathrm{S}^+} \mathsf{Y} \,|\, \mathsf{W}$* – *if there exist $\mathbb{X}$-morphisms s_0, g_1, g_2, h such that s can be decomposed as follows:*



(5)

Compared with Figure 4b, here one allows an extra morphism h to appear after the original superset CI diagrams in Figure 4b; in fact, modulo rewiring, (5) is exactly 

, where $s_1 =$ 

. One intuitive way to think of the extended superset CI is to view the morphisms as certain computational processes [31]: X and Y are independent given W in s if s could be obtained via a computation in which X and Y are computed independently from W (using g_1 and g_2 in (5) respectively), after which some further computation may apply (for which stands the h part in (5)).

▶ **Proposition 35.** *In Markov categories with conditionals, extended superset CI and Markov CI are equivalent. Therefore, in the context of Theorem 32, if $\mathbb{C}$ has conditionals, then the three notions of CI – DIBI CI, Markov CI, and extended superset CI – coincide.*

7 Conclusion

In this paper we provide a general recipe to construct models for DIBI logic, generalising the previously studied probabilistic and relational models. We adopt string diagrams to best visualise the "input-preserving" property that characterises the states in the models, as well as the compositions and subkernel relations, whose definition would be quite convoluted in non-diagrammatic syntax. Then, we derive various new classes of DIBI models of interest. In addition, we define an abstract notion of conditional independence in terms of DIBI formulas. Since our approach can construct DIBI models based on any Markov categories, we are then able to compare the logical CI notion with other definitions of CI proposed in the literature.

There are many promising directions for future work. On the logic side, DIBI logic – interpreted in the probabilistic models – was designed to be the assertion logic of Conditional Probabilistic Separation Logic (CPSL). Our categorical construction of a wide class of DIBI models suggests a generalisation of CPSL to obtain program logics in various scenarios beyond probabilistic programs, in the spirit of Moggi [27].

The notion of CI we propose can be straightforwardly generalised from Markov categories to copy-delete categories (see Section 2). This would allow us to encompass models such as relations with bag semantics in databases [8, 16], sub-probability measures [20]. However, to the best of the authors' knowledge, Proposition 30 fails for generic CD categories. Hence, finding appropriate notions of CI in this more general setting remains an open question.

From a categorical perspective, the definition of the category $\mathbb{C}[\theta]$ deserves further exploration, from at least three angles. First, the $\mathbb{C}[\theta]$-morphisms may be seen as a "bundle" of the images of some syntactic categories of variables and renaming (similar to $\mathbb{S}\mathrm{yn}\mathbb{V}\mathrm{ar}$ from Section 5.3) under suitable functors – usually referred to as "models" in functorial semantics. We would like to make the connection with functorial semantics rigorous in terms of the categorical structures involved [23, 7]. Second, while the current work represents finite sets

of variables using deduplicated finite $\preceq$-ordered lists, towards a more principled treatment, it is worth exploring using nominal string diagrams, a diagrammatic calculus for variables and renaming [2, 3, 1], to represent sets of variables. Third, our categorical treatment of variables seems related to prior work on internalising variables in categories; this problem has been studied since the early days of categorical logic, which led to the construction of polynomial categories [22], later extended to the monoidal setting [29, 17]. It is worth exploring potential connection with this line of work.

References

1 Samuel Balco. *Display calculi and nominal string diagrams*. PhD thesis, University of Leicester, 2020.

2 Samuel Balco and Alexander Kurz. Partially monoidal categories and the algebra of simultaneous substitutions, 2018. Available at `https://gdlyrttnap.pl/resources/papers/syco1.pdf`.

3 Samuel Balco and Alexander Kurz. Nominal String Diagrams. In Markus Roggenbach and Ana Sokolova, editors, *8th Conference on Algebra and Coalgebra in Computer Science (CALCO 2019)*, volume 139 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:20, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.CALCO.2019.18`.

4 Jialu Bao, Simon Docherty, Justin Hsu, and Alexandra Silva. A bunched logic for conditional independence. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–14. IEEE, 2021. `doi:10.1109/LICS52264.2021.9470712`.

5 Gilles Barthe, Justin Hsu, and Kevin Liao. A probabilistic separation logic. *Proceedings of the ACM on Programming Languages*, 4(POPL):1–30, 2019. `doi:10.1145/3371123`.

6 Thomas Bayes. Lii. an essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, f. r. s. communicated by mr. price, in a letter to john canton, a. m. f. r. s. *Philosophical Transactions of the Royal Society of London*, 53:370–418, 1763. `doi:10.1098/rstl.1763.0053`.

7 Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. Deconstructing lawvere with distributive laws. *Journal of logical and algebraic methods in programming*, 95:128–146, 2018. `doi:10.1016/j.jlamp.2017.12.002`.

8 Surajit Chaudhuri and Moshe Y Vardi. Optimization of real conjunctive queries. In *Proceedings of the twelfth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 59–70, 1993. `doi:10.1145/153850.153856`.

9 Kenta Cho and Bart Jacobs. Disintegration and Bayesian inversion via string diagrams. *Mathematical Structures in Computer Science*, 29(7):938–971, 2019. `doi:10.1017/S0960129518000488`.

10 Simon Docherty. *Bunched logics: a uniform approach*. PhD thesis, UCL (University College London), 2019.

11 Brendan Fong and David I Spivak. *An invitation to applied category theory: seven sketches in compositionality*. Cambridge University Press, 2019. `doi:10.1017/9781108668804`.

12 Tobias Fritz. A synthetic approach to markov kernels, conditional independence and theorems on sufficient statistics. *Advances in Mathematics*, 370:107239, August 2020. `doi:10.1016/j.aim.2020.107239`.

13 Dan Geiger and Judea Pearl. Logical and algorithmic properties of conditional independence and graphical models. *The annals of statistics*, 21(4):2001–2021, 1993.

14 Clark Glymour, Kun Zhang, and Peter Spirtes. Review of causal discovery methods based on graphical models. *Frontiers in genetics*, 10:524, 2019. `doi:10.3389/fgene.2019.00524`.

15 Peter V Golubtsov. Monoidal kleisli category as a background for information transformers theory. *Inf. Process.*, 2(1):62–84, 2002.

16 Todd J. Green. *Bag Semantics*, pages 201–206. Springer US, Boston, MA, 2009. doi:10.1007/978-0-387-39940-9_979.

17 C. Hermida and R.D. Tennent. Monoidal indeterminates and categories of possible worlds. *Theoretical Computer Science*, 430:3–22, 2012. Mathematical Foundations of Programming Semantics (MFPS XXV). doi:10.1016/j.tcs.2012.01.001.

18 Martin Hofmann. On the interpretation of type theory in locally cartesian closed categories. In Leszek Pacholski and Jerzy Tiuryn, editors, *Computer Science Logic*, pages 427–441, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg. doi:10.1007/BFb0022273.

19 Bart Jacobs. Semantics of weakening and contraction. *Annals of pure and applied logic*, 69(1):73–106, 1994. doi:10.1016/0168-0072(94)90020-5.

20 Achim Klenke. *Probability theory: a comprehensive course*. Springer Science & Business Media, 2013. doi:10.1007/978-1-84800-048-3.

21 Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

22 Joachim Lambek and Philip J Scott. *Introduction to higher-order categorical logic*, volume 7. Cambridge University Press, 1988. doi:10.5555/7517.

23 F William Lawvere. Functorial semantics of algebraic theories. *Proceedings of the National Academy of Sciences*, 50(5):869–872, 1963. doi:10.1073/pnas.50.5.869.

24 John M Li, Amal Ahmed, and Steven Holtzen. Lilac: a modal separation logic for conditional probability. *Proceedings of the ACM on Programming Languages*, 7(PLDI):148–171, 2023. doi:10.1145/3591226.

25 Pierre Simon marquis de Laplace. *Théorie analytique des probabilités*, volume 7. Courcier, 1820.

26 Ueli Maurer. Abstract models of computation in cryptography. In *Cryptography and Coding: 10th IMA International Conference, Cirencester, UK, December 19-21, 2005. Proceedings 10*, pages 1–12. Springer, 2005.

27 Eugenio Moggi. Notions of computation and monads. *Information and computation*, 93(1):55–92, 1991. doi:10.1016/0890-5401(91)90052-4.

28 Peter W O'Hearn and David J Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, 1999. doi:10.2307/421090.

29 Duško Pavlović. Categorical logic of names and abstraction in action calculi. *Mathematical Structures in Computer Science*, 7(6):619–637, 1997. doi:10.1017/S0960129597002296.

30 Dusko Pavlovic. Chasing diagrams in cryptography. In *Categories and Types in Logic, Language, and Physics: Essays Dedicated to Jim Lambek on the Occasion of His 90th Birthday*, pages 353–367. Springer, 2014.

31 Dusko Pavlovic. *Programs as diagrams: From categorical computability to computable categories*. Springer Nature, 2023. doi:10.1007/978-3-031-34827-3.

32 Judea Pearl. *Causality*. Cambridge university press, 2009. doi:10.1017/CBO9780511803161.

33 Robin Piedeleu and Fabio Zanasi. An introduction to string diagrams for computer scientists. *CoRR*, abs/2305.08768, 2023.

34 David J Pym. *The semantics and proof theory of the logic of bunched implications*, volume 26. Springer Science & Business Media, 2013. doi:10.1007/978-94-017-0091-7.

35 John C Reynolds. Separation logic: A logic for shared mutable data structures. In *Proceedings 17th Annual IEEE Symposium on Logic in Computer Science*, pages 55–74. IEEE, 2002. URL: https://dl.acm.org/doi/10.5555/645683.664578.

36 Peter Selinger. A survey of graphical languages for monoidal categories. In *New structures for physics*, pages 289–355. Springer, 2010. doi:10.1007/978-3-642-12821-9_4.

37 Claude E Shannon. Communication theory of secrecy systems. *The Bell system technical journal*, 28(4):656–715, 1949.

38 Alfred Tarski. *Logic, semantics, metamathematics: papers from 1923 to 1938*. Hackett Publishing, 1983. doi:10.2307/2216869.

A Background on Monads

We first recall the basic definition of monads. We refer to [12, Sect. 3] for an overview of the material in this section. An endofunctor $\mathcal{T}\colon \mathbb{C} \to \mathbb{C}$ is a *monad* if it has a unit $\eta^{\mathcal{T}} : 1_{\mathbb{C}} \to \mathcal{T}$ and a multiplication $\mu^{\mathcal{T}} : \mathcal{T}^2 \to \mathcal{T}$ natural transformations satisfying certain compatibility conditions. Every monad $\mathcal{T}\colon \mathbb{C} \to \mathbb{C}$ induces a Kleisli category $\mathcal{K}\ell(\mathcal{T})$, whose objects are exactly $\mathbb{C}$-objects, and morphisms $\mathsf{X} \to \mathsf{Y}$ are $\mathbb{C}$ morphisms of type $\mathsf{X} \to \mathcal{T}\mathsf{Y}$, with compositions of $f\colon \mathsf{X} \to \mathcal{T}\mathsf{Y}$ and $g\colon \mathsf{Y} \to \mathcal{T}\mathsf{Z}$ given by $\mathsf{X} \xrightarrow{f} \mathcal{T}\mathsf{Y} \xrightarrow{\mathcal{T}g} \mathcal{T}\mathcal{T}\mathsf{Z} \xrightarrow{\mu^{\mathcal{T}}_{\mathsf{Z}}} \mathcal{T}\mathsf{Z}$. We will write the morphisms in $\mathcal{K}\ell(\mathcal{T})$ as $\mathsf{X} \nrightarrow \mathsf{Y}$ to distinguish them from their counterpart $\mathsf{X} \to \mathcal{T}\mathsf{Y}$ in $\mathbb{C}$. Importantly, if $\mathbb{C}$ is a SMC and $\mathcal{T}$ is a commutative monad, then $\mathcal{K}\ell(\mathcal{T})$ is also an SMC [19]. If $\mathcal{T}$ is affine symmetric monoidal, then $\mathcal{K}\ell(\mathcal{T})$ is a Markov category [15, 9].

In the remainder of this section, we recall the monads used in this paper: the distribution monad $\mathcal{D}$, the powerset monad $\mathcal{P}$ (and $\mathcal{P}_i$), the Giry monad $\mathcal{G}$, and the Radon monad $\mathcal{R}$.

▶ **Definition 36** (Discrete Distribution Monad). *The* discrete distribution *monad $\mathcal{D}$ is an endofunctor on* Set*. It maps a countable set X to the set of distributions over X, i.e., the set containing all functions μ over X is satisfying $\sum_{x\in X} \mu(x) = 1$, and maps a function $f : X \to Y$ to $\mathcal{D}(f) : \mathcal{D}(X) \to \mathcal{D}(Y)$, given by $\mathcal{D}(f)(\mu)(y) := \sum_{f(x)=y} \mu(x)$.*

For the monadic structure, define the unit η by $\eta_X(x) := \delta_x$, where δ_x denotes the Dirac distribution on x: for any $y \in X$, we have $\delta_x(y) = 1$ if $y = x$, otherwise $\delta_x(y) = 0$. Further, define bind*: $\mathcal{D}(X) \to (X \to \mathcal{D}(Y)) \to \mathcal{D}(Y)$ by* bind*$(\mu)(f)(y) := \sum_{p\in\mathcal{D}(Y)} \mathcal{D}(f)(\mu)(p) \cdot p(y)$.*

▶ **Definition 37** (Powerset monad). *The* powerset monad *$\mathcal{P}$ is an endofunctor on* Set*. It maps every set to the set of its subsets $\mathcal{P}(X) = \{U \mid U \subseteq X\}$. We define $\eta_X\colon X \to \mathcal{P}(X)$ mapping each $x \in X$ to the singleton $\{x\}$, and* bind*: $\mathcal{P}(X) \to (X \to \mathcal{P}(Y)) \to \mathcal{P}(Y)$ by* bind*$(U)(f) := \cup\{y \mid \exists x \in U.f(x) = y\}$. When restricted to nonempty powersets, the resulting functor $\mathcal{P}_i$ is still a monad, called the* nonempty powerset monad*.*

The next monad is defined on the category $\mathbb{M}\mathbb{e}\mathbb{a}\mathbb{s}$ of measurable spaces, which consists of measurable spaces (A, Σ_A) as objects, and measurable functions as morphisms. $\mathbb{M}\mathbb{e}\mathbb{a}\mathbb{s}$ is a monoidal category, where the monoidal product on objects and morphisms are given by the product of measurable spaces and measurable functions, respectively. In particular, the monoidal unit consists of the singleton measurable space $(\mathbf{1} = \{\bullet\}, \{\varnothing, \mathbf{1}\})$.

▶ **Definition 38** (Giry Monad). *The* giry monad *$\mathcal{G}$ maps a measurable space (X, Σ_X) to another measurable space $(\mathcal{G}(X), \Sigma_{\mathcal{G}(X)})$, where $\mathcal{G}(X)$ is the set of probability measures over X, and the σ-algebra $\Sigma_{\mathcal{G}(X)}$ is the coarsest σ-algebra over $\mathcal{G}(X)$ making the evaluation function $ev_A : \mathcal{G}(X) \to [0, 1]$, defined by $ev_A(\nu) = \nu(A)$, measurable for any $A \in \Sigma_X$. For each measurable function $f : X \mapsto Y$, $\mathcal{G}f : \mathcal{G}X \to \mathcal{G}Y$ is defined by $(\mathcal{G}f)(\nu)(B) = \nu(f^{-1}(B))$ for $B \in \Sigma_Y$. For the monadic structure, define the unit η by $\eta_X(x) = \delta_x$; define the bind operator* bind*$_{X,Y} : \mathcal{G}X \to ((X \to \mathcal{G}Y) \to \mathcal{G}Y)$ by* bind*$(\nu)(f)(B) = \int_X f(X)(B)d\nu$ for $B \in \Sigma_{\mathcal{G}Y}$.*

▶ **Definition 39** (Radon Monad). *The Radon monad $\mathcal{R}$ is a measure monad on the category of compact Hausdorff spaces. If maps a compact Hausdorff space X to the set of Radom measures μ on X such that $\mu(X) \leq 1$. It maps a continuous map between compact Hausdorff spaces $f : X \to Y$ to the push-forward measure $\mathcal{R}(f) : \mathcal{R}X \to \mathcal{R}Y$ given by $\mathcal{D}(f)(\mu)(y) := \mu(f^{-1}(y))$.*

For the monadic structure: we define the unit η to take a point $x \in X$ to the diract distribution δ_x solely supported at x. We also define the bind operator bind*$_{X,Y} : \mathcal{R}X \to ((X \to \mathcal{R}Y) \to \mathcal{R}Y)$ by* bind*$(\nu)(f)(B) = \int_X f(X)(B)d\nu$.*

The category of stochastic processses $\mathbb{Stoch}$ is the Kleisli category of the Giry monad $\mathcal{G}$. It is helpful to explicate its structure.

▶ **Definition 40.** *The symmetric monoidal category of stochastic processses* $\mathbb{Stoch}$ *has the following components:*

- *objects are measurable spaces* (A, Σ_A)*;*
- *morphisms* $(A, \Sigma_A) \to (B, \Sigma_B)$ *are maps* $f \colon \Sigma_B \times A \to [0,1]$ *satisfying: for arbitrary* $T \in \Sigma_B$*,* $f(T, -) \colon A \to [0,1]$ *is measurable, and for arbitrary* $a \in A$*,* $f(-, a) \colon \Sigma_B \to [0,1]$ *is a probability measure;*
- *compositions of* $f \colon (A, \Sigma_A) \to (B, \Sigma_B)$ *and* $g \colon (B, \Sigma_B) \to (C, \Sigma_C)$ *is the map* $g \circ f \colon \Sigma_C \times A \to [0,1]$ *mapping* (U, a) *to* $\int_{b \in B} g(U, b) \cdot f(db, a)$*;*
- $id_{(A,\Sigma_A)}$ *maps* $(S, a) \in \Sigma_A \times A$ *to* 1 *if* $a \in S$*, and to* 0 *if* $a \notin S$*;*
- *the monoidal product* $\otimes$ *acts on objects as* $(A, \Sigma_A) \otimes (B, \Sigma_B) = (A \times B, \Sigma_A \otimes \Sigma_B)$*, where* $\Sigma_A \otimes \Sigma_B$ *is the smallest sigma-algebra containing* $\Sigma_A \times \Sigma_B$*);*
- *the monoidal product* $\otimes$ *acts on morphisms to obtain product measures. That is,* $(U, V) \in \Sigma_B \times \Sigma_D$ *as follows: given* $f \colon (A, \Sigma_A) \to (B, \Sigma_B)$ *and* $g \colon (C, \Sigma_C) \to (D, \Sigma_D)$*,* $f \otimes g \colon \Sigma_B \otimes \Sigma_D \times A \times C \to [0,1]$ *maps* (U, V, a, c) *to* $f(U, a) \cdot g(V, b)$*.*

Representation of Peano Arithmetic in Separation Logic

Sohei Ito
Nagasaki University, Nagasaki, Japan

Makoto Tatsuta[1]
National Institute of Informatics / Sokendai, Tokyo, Japan

Abstract

Separation logic is successful for software verification of heap-manipulating programs. Numbers are necessary to be added to separation logic for verification of practical software where numbers are important. However, properties of the validity such as decidability and complexity for separation logic with numbers have not been fully studied yet. This paper presents the translation of Pi-0-1 formulas in Peano arithmetic to formulas in a small fragment of separation logic with numbers, which consists only of the intuitionistic points-to predicate, 0 and the successor function. Then this paper proves that a formula in Peano arithmetic is valid in the standard model if and only if its translation in this fragment is valid in the standard interpretation. As a corollary, this paper also gives a perspective proof for the undecidability of the validity in this fragment. Since Pi-0-1 formulas can describe consistency of logical systems and non-termination of computations, this result also shows that these properties discussed in Peano arithmetic can also be discussed in such a small fragment of separation logic with numbers.

2012 ACM Subject Classification Theory of computation → Separation logic

Keywords and phrases First order logic, Separation logic, Peano arithmetic, Presburger arithmetic

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.18

Funding *Sohei Ito*: MEXT/JSPS KAKENHI Grant Number JP21K11756, ROIS NII Open Collaborative Research 2023(22FP03)
Makoto Tatsuta: ROIS NII Open Collaborative Research 2023(22FP03)

1 Introduction

Separation logic is successful both theoretically and practically [5, 12] and has been actively studied. It enables us to verify heap-manipulating programs by concise description of memory. Separation logic itself is also interesting theoretically.

When one uses separation logic for verifying software where numbers are important, one often has to extend separation logic by arithmetic. In order for handling numbers, we have to add Peano arithmetic or Presburger arithmetic to separation logic. Then the decidability of the validity in the system of separation logic with arithmetic under the standard interpretation of numbers becomes a question, since the logical system with decidable validity is more appropriate for software verification systems. In particular, since Presburger arithmetic is decidable for the validity, we might expect that some fragment of Presburger arithmetic with some decidable fragment of separation logic is decidable for the validity. In this paper, we will show that it is not the case by proving that the separation logic with only $\hookrightarrow$, 0 and the successor s without any other separation logic constructs or any arithmetical operations can simulate Π^0_1 formulas of Peano arithmetic, and consequently it is undecidable for the validity in the standard interpretation. It was surprising for us that such a weak fragment

[1] Corresponding author

of separation logic becomes such a strong logic even if we only add 0 and s, because the fragment of separation logic with only $\hookrightarrow$ is known to be decidable for the validity in the standard interpretation [4].

Our main theorem in this paper is a representation theorem which states that there is a translation of Π^0_1 formulas in Peano arithmetic to formulas in the separation logic with only the intuitionistic points-to predicate and numbers such that a formula in Peano arithmetic is valid in the standard model if and only if its translation in the separation logic with numbers is valid in the standard interpretation. Namely, the translation preserves the validity and the non-validity. The undecidability result is obtained as a corollary of this theorem.

The main technique of the proof of our representation theorem is to have an operation table for addition, multiplication, and inequality in a heap, and to remove $x + y = z$, $x \times y = z$ and $x \leq y$ by referring to the value z or the truth of $x \leq y$. For this, we will code the operation table by the consecutive heap cells that contain $0, x + 3, y + 3, x + y + 3$ or $1, x + 3, y + 3, x \times y + 3$ or $2, x + 3, y + 3$ where 3 is an offset and $0, 1$ and 2 are tags for addition, multiplication, and inequality, respectively. In order to define the translation, we will introduce normal form of a bounded formula of Peano arithmetic.

Our translation can be extended to arbitrary formulas of Peano arithmetic, but the representation theorem does not hold for formulas beyond Π^0_1. We will give some Σ^0_1 formula as a counterexample.

Our result shows that discussion about properties described by Π^0_1 formulas such as consistency of logical systems and strong normalization properties for reduction systems in Peano arithmetic can be simulated in the separation logic with numbers. The undecidability of the validity in the separation logic with numbers itself can be proved in a simpler way, by using a similar idea to [7]. We will also give a proof in that way.

There are some undecidability results about the validity of separation logic and some of them use some translations. The separation logic with the 1-field points-to predicate and the separating implication is known to be undecidable for its validity [4]. The separation logic with the 2-field points-to predicate is also known to be undecidable for its validity [7]. It is proved by translating formulas in a first-order logic with one binary relation into the separation logic with the 2-field points-to predicate. On the other hand, there are some decidability results about the validity of separation logic. The separation logic with the 1-field points-to predicate and without the separating implication is known to be decidable for its validity [4]. The quantifier-free separation logic is known to be decidable [6]. It is proved by translating the separation logic into a first-order logic with empty signature. We do not know any work on a translation from some fragment of arithmetic into such a weak separation logic with only $\{\hookrightarrow, 0, s\}$.

If we restrict ourselves to symbolic heaps in separation logic with arithmetic or inductive definitions, there are some decidability results. Symbolic heap entailment with Presburger arithmetic [15], bounded-treewidth symbolic heap entailment [10], symbolic heap entailment with cone inductive definitions [16, 11], and symbolic heap entailment with lists [2, 3, 8, 1] are known to be decidable for their validity. Even for symbolic heaps, if we weaken conditions, they easily become undecidable. Symbolic heap entailment with unrestricted inductive definitions [10], and symbolic heap entailment with bounded-treewidth inductive definitions and implicit existentials [14] are known to be undecidable for their validity.

This paper is organized as follows: Section 2 defines Peano arithmetic. Section 3 defines the separation logic with numbers SLN. Section 4 defines the translation of normal formulas in Peano arithmetic to formulas in SLN and shows the preservation property of the translation. In Section 5 we define an auxiliary translation to normal form and prove the main theorem for the preservation of the translation for Π^0_1 formulas from PA to SLN. Section 6 gives another proof of undecidability for the validity of SLN. Section 7 concludes.

2 Peano arithmetic

In this section, we define Peano arithmetic PA and its standard model.

Let Vars $= \{x, y, \ldots\}$ be the set of variables. The *terms of* PA are defined by:

$$t ::= x \mid 0 \mid s(t) \mid t + t \mid t \times t.$$

The *formulas of* PA are defined by:

$$A ::= t = t \mid t \leq t \mid \neg A \mid A \wedge A \mid A \vee A \mid \exists x A \mid \forall x A.$$

We will write $A \to B$ for $\neg A \vee B$.

We write $s^n(t)$ for $\overbrace{s(\ldots(s}^{n}(t))\ldots)$. We use the abbreviation $\overline{n} = s^n(0)$. We write $A[x := t]$ for the formula obtained by capture-free substitution of t for x in A.

Let $\mathcal{N}$ be the *standard model* of PA, namely, its universe $|\mathcal{N}|$ is $\mathbb{N} = \{0, 1, 2, \ldots\}$, $0^{\mathcal{N}}=0$, $s^{\mathcal{N}}(x) = x + 1$, $+^{\mathcal{N}}(x, y) = x + y$, $\times^{\mathcal{N}}(x, y) = x \times y$, $(\leq)^{\mathcal{N}}(x, y)$ iff $x \leq y$. Let $\sigma : \text{Vars} \to \mathbb{N}$ be a *variable assignment*. We extend σ to terms in a usual way. We write $\sigma[x := n]$ for the variable assignment that assigns n to x and $\sigma(y)$ to y other than x.

We write $\sigma \models A$ when A is true in $\mathcal{N}$ under the variable assignment σ. This relation is defined in a usual way. If $\sigma \models A$ for every variable assignment σ, A is defined to be *valid*. If A does not contain free variables, A is called *closed*.

A formula $\forall x \leq t.A$ is an abbreviation of $\forall x(x \leq t \to A)$, where t does not contain x. A formula $\exists x \leq t.A$ is an abbreviation of $\exists x(x \leq t \wedge A)$, where t does not contain x. We call $\forall x \leq t$ and $\exists x \leq t$ *bounded quantifiers*. A formula A is defined to be *bounded* if every quantifier in A is bounded. If $A \equiv \forall x B$ and B is bounded, A is called a Π^0_1 *formula*.

3 Separation logic with numbers

In this section, we define a small fragment SLN of separation logic with numbers. We will also define the standard interpretation of SLN.

Let Vars $= \{x, y, \ldots\}$ be the set of variables. The *terms of* SLN are defined by:

$$t ::= x \mid 0 \mid s(t).$$

The *formulas of* SLN are defined by:

$$A ::= t = t \mid t \hookrightarrow t \mid \neg A \mid A \wedge A \mid A \vee A \mid \exists x A \mid \forall x A.$$

We will write $A \to B$ for $\neg A \vee B$.

The predicate $t_1 \hookrightarrow t_2$ is the intuitionistic points-to predicate and means that there is some cell of address t_1 which contains t_2 in the heap.

We use the same abbreviation $\overline{n}$ and substitution $A[x := t]$ as in PA. For simplicity, we write $(t \hookrightarrow t_1, \ldots, t_n)$ for $t \hookrightarrow t_1 \wedge \ldots \wedge s^{n-1}(t) \hookrightarrow t_n$. We sometimes write only one quantifier for consecutive quantifiers in a usual way like $\forall xy \exists zw$ for $\forall x \forall y \exists z \exists w$.

Now we define the *standard interpretation* $[\![\cdot]\!]$ of SLN. Let $\mathbb{N}$ be $\{0, 1, \ldots\}$. We use $\mathbb{N}$ for both the sets of addresses and values. Let $[\![0]\!] = 0, [\![s]\!](x) = x + 1$. Let $\sigma : \text{Vars} \to \mathbb{N}$ be a *variable assignment*. The extension of σ to terms and the variable assignment $\sigma[x := n]$ are defined similarly to those in PA. A heap is a finite function $h : \mathbb{N} \to_{\text{fin}} \mathbb{N}$. A heap represents a state of the memory.

For a formula A of SLN, we define $\sigma, h \models A$ by:

$$\begin{array}{lll}
\sigma, h \models t_1 = t_2 & \text{iff} & \sigma(t_1) = \sigma(t_2), \\
\sigma, h \models t_1 \hookrightarrow t_2 & \text{iff} & h(\sigma(t_1)) = \sigma(t_2), \\
\sigma, h \models \neg A & \text{iff} & \sigma, h \not\models A, \\
\sigma, h \models A_1 \wedge A_2 & \text{iff} & \sigma, h \models A_1 \text{ and } \sigma, h \models A_2, \\
\sigma, h \models A_1 \vee A_2 & \text{iff} & \sigma, h \models A_1 \text{ or } \sigma, h \models A_2, \\
\sigma, h \models \exists x A & \text{iff} & \text{for some } n \in \mathbb{N}, \sigma[x := n], h \models A, \\
\sigma, h \models \forall x A & \text{iff} & \text{for all } n \in \mathbb{N}, \sigma[x := n], h \models A.
\end{array}$$

$\sigma, h \models A$ means that A is true under the variable assignment σ and the heap h. A formula A is defined to be *valid* if $\sigma, h \models A$ for all σ and h. If a formula does not contain atoms $t \hookrightarrow u$, the truth of the formula does not depend on heaps.

The validity defined in this section is the validity in the standard interpretation of SLN, and it is different from the ordinary validity for separation logic, since the interpretations depend on the set of addresses in the definition of the ordinary validity.

4 Translation of Normal Formulas in PA into SLN

In this section, we define the translation $(\cdot)^\circ$ of bounded formulas in PA to formulas in SLN, and prove that the translation preserves the validity and the non-validity.

The key of the translation is to keep an operation table for addition, multiplication and inequality in a heap, and a resulting formula in SLN refers to the table instead of using the addition, multiplication and inequality symbols. To state that a heap keeps the operation table, we will use a table heap condition. For proving the preservation of the translation, we will use a simple table heap, which is a heap that contains all the operation entries of some size. Since the table in a heap is finite, to estimate the necessary size of the operation table for translating a given formula, we will use the upper bound of arguments in the formula.

We will first define normal form of a bounded formula in PA, which we will translate into a formula in SLN. Next we will define a table heap condition, which guarantees that a heap has an operation table for addition, multiplication and inequality. Then we will define the translation of a normal formula in PA into a formula in SLN. Then we will define a simple table heap and the upper bound of arguments in a formula. Finally we will prove the preservation of the translation.

We write $\exists(x = t)A$ for an abbreviation of $\exists x(x = t \wedge A)$, where t does not contain x.

Our translation is defined for only normal formulas. This does not lose the generality since any bounded formula can be transformed into a normal formula in Section 5. In a normal formula, $+$ and $\times$ appear only in t of $\exists(x = t)$. Moreover, this t is of the form $a + b$ or $a \times b$ where a, b do not contain $+$ or $\times$.

▶ **Definition 4.1** (Normal form). *Normal forms* of PA are given by A in the following grammar:

$$A ::= B \mid \forall x \leq t.A \mid \exists x \leq t.A \mid \exists(x = t)A$$

satisfying the following conditions: (1) B is a disjunctive normal form of a formula in PA without quantifiers, $+$, $\times$, and formulas of the form $\neg(t \leq u)$, (2) each t in $\forall x \leq t$ and $\exists x \leq t$ does not contain $+$, $\times$, and (3) each t in $\exists(x = t)$ is of the form $a + b$ or $a \times b$ for some terms a and b that do not contain $+$ or $\times$.

The table heap condition is defined as the formula H in the next definition. It guarantees that a heap that satisfies H contains a correct operation table for $+$, $\times$ and $\leq$. The formulas Add, Mult and Ineq in the following definition refer to the operation table when a heap satisfies

the table heap condition. The normal formula enables us to represent each occurrence of $+$, $\times$ and $\leq$ by Add, Mult and Ineq, respectively. We will write $[t]$ for $s^3(t)$ for readability, since the offset is 3.

▶ **Definition 4.2** (Table Heap Condition). H, $\mathsf{Add}(x,y,z)$, $\mathsf{Mult}(x,y,z)$ and $\mathsf{Ineq}(x,y)$ are the formulas defined by:

$$
\begin{aligned}
H_{\mathsf{Add1}} &\equiv \forall ay((a \hookrightarrow 0,[0],[y]) \to s^3(a) \hookrightarrow [y]),\\
H_{\mathsf{Add2}} &\equiv \forall axy((a \hookrightarrow 0,[s(x)],[y])\\
&\qquad \to \exists bz((b \hookrightarrow 0,[x],[y],[z]) \wedge s^3(a) \hookrightarrow [s(z)])),\\
H_{\mathsf{Mult1}} &\equiv \forall ay((a \hookrightarrow \overline{1},[0],[y]) \to s^3(a) \hookrightarrow [0]),\\
H_{\mathsf{Mult2}} &\equiv \forall axy((a \hookrightarrow \overline{1},[s(x)],[y]) \to \exists bz((b \hookrightarrow \overline{1},[x],[y],[z]) \wedge\\
&\qquad \exists cw((c \hookrightarrow 0,[z],[y],[w]) \wedge s^3(a) \hookrightarrow [w]))),\\
H_{\mathsf{Ineq1}} &\equiv \forall axy((a \hookrightarrow \overline{2},[s(x)],[y]) \to \exists zb(y = s(z) \wedge (b \hookrightarrow \overline{2},[x],[z]))),\\
H_{\mathsf{Ineq2}} &\equiv \forall axy((a \hookrightarrow \overline{2},[s(x)],[y]) \to \exists b(b \hookrightarrow \overline{2},[x],[y])),\\
H &\equiv H_{\mathsf{Add1}} \wedge H_{\mathsf{Add2}} \wedge H_{\mathsf{Mult1}} \wedge H_{\mathsf{Mult2}} \wedge H_{\mathsf{Ineq1}} \wedge H_{\mathsf{Ineq2}},\\
\mathsf{Add}(x,y,z) &\equiv \forall a((a \hookrightarrow 0,[x],[y]) \to s^3(a) \hookrightarrow [z]),\\
\mathsf{Mult}(x,y,z) &\equiv \forall a((a \hookrightarrow \overline{1},[x],[y]) \to s^3(a) \hookrightarrow [z]),\\
\mathsf{Ineq}(x,y) &\equiv \exists a(a \hookrightarrow \overline{2},[x],[y]).
\end{aligned}
$$

The formula H forces a heap to have a table that contains results of addition, multiplication and inequality for some natural numbers. Each entry for addition and multiplication consists of four cells, and each entry for inequality consists of three cells. If the first cell contains 0, then the entry is for addition. If the first cell contains 1, then the entry is for multiplication. If the first cell contains 2, then the entry is for inequality. The second and third cells of an entry represent arguments of addition, multiplication or inequality. The entries for $+, \times$ have the forth cells, which contain the results of addition or multiplication. For inequality, if there is an entry for two arguments x and y, then $x \leq y$ holds. Since 0, 1 and 2 have a special meaning, arguments and results are stored by adding three. The definition of H uses the following inductive definitions of addition and multiplication: $s(x) + y = s(x + y)$ and $(x + 1) \times y = x \times y + x$. The formulas H_{Add1} and H_{Mult1} force the base cases of addition and multiplication, respectively, and H_{Add2} and H_{Mult2} force the induction steps of addition and multiplication, respectively. H_{Ineq1} means that if there is an entry for $x + 1 \leq y$, then the entry for $x \leq y - 1$ exists in the heap. H_{Ineq2} means that if there is an entry for $x + 1 \leq y$, then the entry for $x \leq y$ exists in the heap.

We will show that the formula H actually forces the heap to have a correct table for addition, multiplication and inequality (the claims (1), (2) and (3) below). The claim (4) below says that H ensures that if a heap contains an entry for $u \leq u$, then it contains all the entries for $t \leq u$.

▶ **Lemma 4.3.** *Let σ be a variable assignment and h be a heap.*

(1) *If $\sigma, h \models H$, $h(m) = 0$, $h(m+1) = n+3$ and $h(m+2) = k+3$, then $h(m+3) = n+k+3$.*

(2) *If $\sigma, h \models H$, $h(m) = 1$, $h(m+1) = n+3$ and $h(m+2) = k+3$, then $h(m+3) = n \times k+3$.*

(3) *If $\sigma, h \models H$, $h(m) = 2$, $h(m+1) = n+3$, $h(m+2) = k+3$, then $n \leq k$.*

(4) *If $\sigma, h \models H$, $\sigma(t) \leq \sigma(u)$, $\sigma, h \models \mathsf{Ineq}(u,u)$, then $\sigma, h \models \mathsf{Ineq}(t,u)$.*

Proof.

(1) We will show the claim by induction on n.
(Base case) Let $n = 0$. Since $h(m) = 0$, $h(m+1) = 3$, $h(m+2) = k+3$ and $\sigma, h \models H_{\mathsf{Add1}}$, we have $\sigma[a := m, x := n, y := k], h \models s^3(a) \hookrightarrow [y]$. Hence, we have $h(m+3) = \sigma[a := m, x := n, y := k]([y]) = k+3 = n+k+3$.
(Induction step) Let $n > 0$. Then, $n-1 \geq 0$. Let $\sigma' = \sigma[a := m, x := n-1, y := k]$. Since $h(m) = 0$, $h(m+1) = n+3$, $h(m+2) = k+3$ and $\sigma, h \models H_{\mathsf{Add2}}$, we have $\sigma', h \models \exists bz((b \hookrightarrow 0, [x], [y], [z]) \wedge s^3(a) \hookrightarrow [s(z)])$. There exist q and ℓ such that $\sigma'[b := q, z := \ell], h \models (b \hookrightarrow 0, [x], [y], [z]) \wedge s^3(a) \hookrightarrow [s(z)]$. That is, $h(q) = 0$, $h(q+1) = (n-1)+3$, $h(q+2) = k+3$, $h(q+3) = \ell+3$ and $h(m+3) = \ell+4$. By induction hypothesis, we have $h(q+3) = (n-1)+k+3 = n+k+2$. That is, $\ell = n+k-1$. Thus, we have $h(m+3) = \ell+4 = n+k-1+4 = n+k+3$.

(2) We will show the claim by induction on n.
(Base case) Let $n = 0$. Since $h(m) = 1$, $h(m+1) = 3$, $h(m+2) = k+3$ and $\sigma, h \models H_{\mathsf{Mult1}}$, we have $\sigma[a := m, x := n, y := k], h \models s^3(a) \hookrightarrow [0]$. Hence, we have $h(m+3) = \sigma[a := m, x := n, y := k]([0]) = 3 = n \times k + 3$.
(Induction step) Let $n > 0$. Then, $n-1 \geq 0$. Let $\sigma' = \sigma[a := m, x := n-1, y := k]$. Since $h(m) = 1$, $h(m+1) = n+3$, $h(m+2) = k+3$ and $\sigma, h \models H_{\mathsf{Mult2}}$, we have $\sigma', h \models \exists bz((b \hookrightarrow \overline{1}, [x], [y], [z]) \wedge \exists cw((c \hookrightarrow 0, [z], [y], [w]) \wedge s^3(a) \hookrightarrow [w]))$. There exist q and ℓ such that $\sigma'[b := q, z := \ell], h \models (b \hookrightarrow \overline{1}, [x], [y], [z]) \wedge \exists cw((c \hookrightarrow 0, [z], [y], [w]) \wedge s^3(a) \hookrightarrow [w])$. That is, $h(q) = 1, h(q+1) = (n-1)+3, h(q+2) = k+3, h(q+3) = \ell+3$. So by induction hypothesis, $h(q+3) = (n-1) \times k + 3$. Thus $\ell = (n-1) \times k$. Furthermore, since $\sigma'[b := q, z := \ell], h \models \exists cw((c \hookrightarrow 0, [z], [y], [w]) \wedge s^3(a) \hookrightarrow [w])$, we have $\sigma'[b := q, z := \ell, c := r, w := p], h \models (c \hookrightarrow 0, [z], [y], [w]) \wedge s^3(a) \hookrightarrow [w]$ for some r and p. That is, $h(r) = 0, h(r+1) = \ell+3, h(r+2) = k+3, h(r+3) = p+3$ and $h(m+3) = p+3$. By (1) of this Lemma, we have $p = \ell + k$. With this and $\ell = (n-1) \times k$, we have $p = (n-1) \times k + k = n \times k$. Hence, $h(m+3) = p+3 = n \times k + 3$.

(3) We will show the claim by induction on n.
(Base case) Let $n = 0$. We immediately have $n \leq k$.
(Induction step) Let $n > 0$. Then, $n-1 \geq 0$. Let $\sigma' = \sigma[a := m, x := n-1, y := k]$. Since $h(m) = 2$, $h(m+1) = n+3$, $h(m+2) = k+3$ and $\sigma, h \models H_{\mathsf{Ineq1}}$, we have $\sigma', h \models \exists zb(y = s(z) \wedge (b \hookrightarrow \overline{2}, [x], [z]))$. Thus, there exist ℓ and q such that $\sigma'[z := \ell, b := p], h \models y = s(z) \wedge (b \hookrightarrow \overline{2}, [x], [z])$, that is, $h(p) = 2$, $h(p+1) = (n-1)+3$, $h(p+2) = \ell+3 = (k-1)+3$. By induction hypothesis, $n-1 \leq k-1$, that is, $n \leq k$.

(4) We will show the claim by induction on $\sigma(u) - \sigma(t)$.
(Base case) Let $\sigma(u) - \sigma(t) = 0$, i.e. $\sigma(t) = \sigma(u)$. By assumption, we have $\sigma, h \models \mathsf{Ineq}(u, u)$. Since $\sigma(t) = \sigma(u)$, we have the claim.
(Induction step) Let $\sigma(u) - \sigma(t) > 0$, i.e. $\sigma(t) < \sigma(u)$. Since $\sigma(u) - (\sigma(t)+1) < \sigma(u) - \sigma(t)$, we have $\sigma, h \models \mathsf{Ineq}(s(t), u)$ by induction hypothesis. That is, $\sigma, h \models \exists a(a \hookrightarrow \overline{2}, [s(t)], [u])$. Since $\sigma, h \models H_{\mathsf{Ineq2}}$, we have $\sigma, h \models \exists b(b \hookrightarrow \overline{2}, [t], [u])$, that is, $\sigma, h \models \mathsf{Ineq}(t, u)$. ◀

Now we define the translation of normal formulas in PA into formulas in SLN. In the translation, $+$, $\times$ and $\leq$ are replaced by Add, Mult and Ineq with the table heap condition.

▶ **Definition 4.4** (Translation $(\cdot)^\circ$)**.** Let A be a normal formula in PA. We define SLN formula $(\forall x A)^\circ$ as:

$$\begin{aligned}
B^\circ &\equiv B^{\leq} \text{ if } B \text{ is quantifier-free,}\\
(\exists x \leq t.B)^\circ &\equiv H \to \neg \mathsf{Ineq}(t,t) \vee \exists x(\mathsf{Ineq}(x,t) \wedge B^\circ),\\
(\forall x \leq t.B)^\circ &\equiv H \to \forall x(\neg \mathsf{Ineq}(x,t) \vee B^\circ),\\
(\exists (x = t+u)B)^\circ &\equiv H \to \exists x(\mathsf{Add}(t,u,x) \wedge B^\circ),\\
(\exists (x = t\times u)B)^\circ &\equiv H \to \exists x(\mathsf{Mult}(t,u,x) \wedge B^\circ),\\
(\forall x A)^\circ &\equiv \forall x A^\circ,
\end{aligned}$$

where $B^{\leq}$ is obtained from B by replacing each positive occurrence of $t \leq u$ by $H \to \neg\mathsf{Ineq}(u,t) \vee t = u$.

Example. For a normal formula

$$\begin{aligned}
A \equiv &\exists (x_1 = x + s(x))\exists (x_2 = x + x_1)\forall y \leq x_2.\\
&\exists (x_3 = x + y)\exists (x_4 = y \times x_3)\exists (x_5 = x + x_4)(0 \leq x_5),
\end{aligned}$$

its translation A° is

$$\begin{aligned}
A^\circ \equiv &H \to \exists x_1(\mathsf{Add}(x, s(x), x_1) \wedge (H \to \exists x_2(\mathsf{Add}(x, x_1, x_2) \wedge (H \to \forall y(\neg \mathsf{Ineq}(y, x_2) \vee\\
&(H \to \exists x_3(\mathsf{Add}(x, y, x_3) \wedge (H \to \exists x_4(\mathsf{Mult}(y, x_3, x_4) \wedge\\
&(H \to \exists x_5(\mathsf{Add}(x, x_4, x_5) \wedge (H \to \neg \mathsf{Ineq}(x_5, 0) \vee 0 = x_5)))))))))))).
\end{aligned}$$

Our goal is to show that for any Π^0_1 formula A of PA, A is valid in PA if and only if A° is valid in SLN. Therefore, A° should hold for *every* heap h. By the definition of $(\cdot)^\circ$, $x = t+u$ and $x = t\times u$ are translated into $H \to \mathsf{Add}(t,u,x)$ and $H \to \mathsf{Mult}(t,u,x)$, respectively. Thus, if a heap h does not have a sufficiently large table for $x = t+u$ and $x = t\times u$, the translated formulas are trivially true. Since we demand that A° hold for all heaps, there is h that contains a sufficiently large table. Furthermore, if the addition and multiplication in the formula are correct in such a sufficiently large heap, they must be correct in every heap, because addition and multiplication are numeric properties and do not depend on heaps. The same is true for inequality. This is the key idea to prove our goal. That is, $\sigma \models A$ if and only if $\sigma, h \models A^\circ$ for *sufficiently large* h if and only if $\sigma, h \models A^\circ$ for *all* h. We will prove them in Lemmas 4.10 and 4.11 later.

Since we demand that A° hold for all heaps, we define the translation of $t \leq u$ to be $H \to \neg\mathsf{Ineq}(u,t) \vee t = u$ and we do not straightforwardly define it to be $H \to \mathsf{Ineq}(t,u)$, because $\mathsf{Ineq}(t,u)$ demands the heap to contain the entry for $t \leq u$, which is not possible if the heap is not sufficiently large. Furthermore, the translation of $\exists x \leq t.B$ is not simply $H \to \exists x(\mathsf{Ineq}(x,t) \wedge B)$ but rather seemingly tricky $H \to \neg\mathsf{Ineq}(t,t) \vee \exists x(\mathsf{Ineq}(x,t) \wedge B^\circ)$. If we adopt the simple translation, we may not be able to find x such that the entry for $x \leq t$ is in the heap when it is not sufficiently large. Our idea is to let such a case be true. Therefore, we allow the case $\neg\mathsf{Ineq}(t,t)$, which is true if the heap may not contain some entries for $\cdot \leq t$.

First, we estimate the necessary size of the operation table for a given formula and a given variable assignment. This size is defined in the next definition.

▶ **Definition 4.5.** Let A be prenex and disjunctive normal form of a bounded formula in PA and σ be a variable assignment. We define the number $\max(\sigma, A)$ by:

$$\begin{aligned}
&\max(\sigma, t \le u) = \max\{\sigma(t), \sigma(u)\},\\
&\max(\sigma, t = u) = \{0\},\\
&\max(\sigma, \neg B) = \max(\sigma, B)\\
&\max(\sigma, B \wedge C) = \max\{\max(\sigma, B), \max(\sigma, C)\}\\
&\max(\sigma, B \vee C) = \max\{\max(\sigma, B), \max(\sigma, C)\}\\
&\max(\sigma, \forall x \le t.B) = \max\{\sigma(t), \max(\sigma, B[x := t])\},\\
&\max(\sigma, \exists x \le t.B) = \max\{\sigma(t), \max(\sigma, B[x := t])\},\\
&\max(\sigma, \exists (x = t)B) = \max\{\sigma(t), \max(\sigma, B[x := t])\}.
\end{aligned}$$

Example. By using the above definition one by one, we have

$$\begin{aligned}
&\max(\sigma, \exists(x_1 = x + s(x))\exists(x_2 = x + x_1)\forall y \le x_2.\\
&\quad \exists(x_3 = x + y)\exists(x_4 = y \times x_3)\exists(x_5 = x + x_4)(0 \le x_5))\\
&= \max\{\sigma(x + s(x)), \max(\sigma, \exists(x_2 = x + (x + s(x)))\forall y \le x_2.\\
&\quad \exists(x_3 = x + y)\exists(x_4 = y \times x_3)\exists(x_5 = x + x_4)(0 \le x_5))\}\\
&= \max\{\sigma(x + s(x)), \max\{\sigma(x + (x + s(x))), \max(\sigma, \forall y \le x + (x + s(x)).\\
&\quad \exists(x_3 = x + y)\exists(x_4 = y \times x_3)\exists(x_5 = x + x_4)(0 \le x_5))\}\}\\
&= \max\{\sigma(x + s(x)), \max\{\sigma(x + (x + s(x))), \max\{\sigma(x + (x + s(x))),\\
&\quad \max(\sigma, \exists(x_3 = x + (x + (x + s(x))))\exists(x_4 = (x + (x + s(x))) \times x_3)\\
&\quad \exists(x_5 = x + x_4)(0 \le x_5))\}\}\}\\
&= \max\{\sigma(x + s(x)), \max\{\sigma(x + (x + s(x))), \max\{\sigma(x + (x + s(x))),\\
&\quad \max\{\sigma(x + (x + (x + s(x)))),\\
&\quad \max(\sigma, \exists(x_4 = (x + (x + s(x))) \times (x + (x + (x + s(x)))))\exists(x_5 = x + x_4)(0 \le x_5))\}\}\}\}\\
&= \max\{\sigma(x + s(x)), \max\{\sigma(x + (x + s(x))), \max\{\sigma(x + (x + s(x))),\\
&\quad \max\{\sigma(x + (x + (x + s(x)))),\\
&\quad \max\{\sigma((x + (x + s(x))) \times (x + (x + (x + s(x))))),\\
&\quad \max(\sigma, \exists(x_5 = x + (x + (x + s(x))) \times (x + (x + (x + s(x)))))(0 \le x_5))\}\}\}\}\}\\
&= \max\{\sigma(x + s(x)), \max\{\sigma(x + (x + s(x))), \max\{\sigma(x + (x + s(x))),\\
&\quad \max\{\sigma(x + (x + (x + s(x)))),\\
&\quad \max\{\sigma((x + (x + s(x)) \times (x + (x + (x + s(x))))),\\
&\quad \max\{\sigma(x + (x + (x + s(x))) \times (x + (x + (x + s(x))))),\\
&\quad \max(\sigma, 0 \le x + (x + (x + s(x))) \times (x + (x + (x + s(x)))))\}\}\}\}\}\\
&= \max\{\sigma(x + s(x)), \max\{\sigma(x + (x + s(x))), \max\{\sigma(x + (x + s(x))),\\
&\quad \max\{\sigma(x + (x + (x + s(x)))),\\
&\quad \max\{\sigma((x + s(x)) \times (x + (x + (x + s(x))))),\\
&\quad \max\{\sigma(x + (x + s(x)) \times (x + (x + (x + s(x))))),\\
&\quad \max\{0, \sigma(x + (x + (x + s(x))) \times (x + (x + (x + s(x)))))\}\}\}\}\}\}\\
&= \sigma(x) + (3\sigma(x) + 1)(4\sigma(x) + 1).
\end{aligned}$$

Next, for a given size n we will define a heap that covers addition of arguments $\le n^2$ and multiplication and inequality of arguments $\le n$. We call it a *simple table heap*.

▶ **Definition 4.6.** For a number n, we define a heap h_n as the heap defined by:

$$h_n(x) = \begin{cases} 0 & (x = 4i, i < (n^2+1)^2) \\ i \bmod (n^2+1) + 3 & (x = 4i+1, i < (n^2+1)^2) \\ \lfloor i/(n^2+1) \rfloor + 3 & (x = 4i+2, i < (n^2+1)^2) \\ h_n(x-2) + h_n(x-1) - 3 & (x = 4i+3, i < (n^2+1)^2) \\ 1 & (x = c_1 + 4i, i < (n+1)^2) \\ i \bmod (n+1) + 3 & (x = c_1 + 4i + 1, i < (n+1)^2) \\ \lfloor i/(n+1) \rfloor + 3 & (x = c_1 + 4i + 2, i < (n+1)^2) \\ (h_n(x-2) - 3) \times (h_n(x-1) - 3) + 3 & (x = c_1 + 4i + 3, i < (n+1)^2) \\ 2 & (x = c_2 + 3i, i < (n+1)^2) \\ i \bmod (n+1) + 3 & (x = c_2 + 3i + 1, i < (n+1)^2) \\ n + 3 & (x = c_2 + 3i + 2, i < (n+1)^2, \\ & \quad \lfloor i/(n+1) \rfloor < i \bmod (n+1)) \\ \lfloor i/(n+1) \rfloor + 3 & (x = c_2 + 3i + 2, i < (n+1)^2, \\ & \quad \lfloor i/(n+1) \rfloor \geq i \bmod (n+1)) \\ \text{undefined} & \text{otherwise} \end{cases}$$

where $c_1 = 4(n^2+1)^2$ and $c_2 = c_1 + 4(n+1)^2$.

h_n has the operation table that has entries of $+$ for arguments $\leq n^2$ and the entries of $\times$ and $\leq$ for arguments $\leq n$. The i-th entry for $+$ contains the result of addition of $x = i \bmod (n^2+1)$ and $y = \lfloor i/(n^2+1) \rfloor$, that is, $h(4i) = 0$, $h(4i+1) = x+3$, $h(4i+2) = y+3$ and $h(4i+3) = x+y+3$. The i-th entry for $\times$ contains the result of multiplication of $x = i \bmod (n+1)$ and $y = \lfloor i/(n+1) \rfloor$, that is, $h(c_1 + 4i) = 0$, $h(c_1 + 4i + 1) = x + 3$, $h(c_1 + 4i + 2) = y + 3$ and $h(c_1 + 4i + 3) = x \times y + 3$. The i-th entry for $\leq$ signifies inequality of $x = i \bmod (n+1)$ and $y = \lfloor i/(n+1) \rfloor$ or $y = n$, where $h(c_2 + 4i) = 2$, $h(c_2 + 4i + 1) = x + 3$, and $h(c_2 + 4i + 2) = y + 3$ if $x \leq y$ and $h(c_2 + 4i + 2) = n + 3$ if $x > y$.

The next lemma shows that the simple table heap h_n satisfies the table heap condition H.

▶ **Lemma 4.7.** *For a variable assignment σ, we have $\sigma, h_n \models H$.*

Proof. h_n clearly satisfies H. ◀

The next lemma shows that the truth of $t + u = v$, $t \times u = v$ and $t \leq u$ in PA for the standard model is equivalent to the truth of their translations in SLN for the standard interpretation for the simple table heap.

▶ **Lemma 4.8.** *For $n \geq \max\{\sigma(t), \sigma(u)\}$, the following hold.*

(1) *$\sigma \models t + u = v$ if and only if $\sigma, h_n \models \mathsf{Add}(t, u, v)$.*

(2) *$\sigma \models t \times u = v$ if and only if $\sigma, h_n \models \mathsf{Mult}(t, u, v)$.*

(3) *$\sigma \models t \leq u$ if and only if $\sigma, h_n \models \mathsf{Ineq}(t, u)$.*

Proof.

(1) Only-if-direction: Since $n \geq \max\{\sigma(t), \sigma(u)\}$, by the definition of h_n, there exists p such that $\sigma[a := p], h_n \models (a \hookrightarrow 0, [t], [u])$. That is, $h_n(p) = 0, h_n(p+1) = \sigma(t) + 3, h_n(p+2) = \sigma(u) + 3$. By the definition of h_n, we have $h_n(p+3) = \sigma(t) + \sigma(u) + 3$. By assumption, $\sigma(t) + \sigma(u) = \sigma(v)$. Therefore, $h_n(p+3) = \sigma(v) + 3$. Thus, $\sigma[a := p], h_n \models s^3(a) \hookrightarrow [v]$. Hence, $\sigma, h_n \models \mathsf{Add}(t, u, v)$.

If-direction: Since $n \geq \max\{\sigma(t), \sigma(u)\}$, by the definition of h_n, there exists p such that $\sigma[a := p], h_n \models (a \hookrightarrow 0, [t], [u], [v])$. Thus, $h_n(p) = 0, h_n(p+1) = \sigma(t) + 3, h_n(p+2) = \sigma(u) + 3, h_n(p+3) = \sigma(v) + 3$. By the definition of h_n, we have $h_n(p+3) = (h_n(p+1) - 3) + (h_n(p+2) - 3) + 3$. Since $h_n(p+1) = \sigma(t) + 3$ and $h_n(p+2) = \sigma(u) + 3$, we have $h_n(p+3) = \sigma(t) + \sigma(u) + 3$. Thus, we have $\sigma(t) + \sigma(u) = \sigma(v)$. Hence, $\sigma \models t + u = v$.

(2) The claim can be shown similarly to (1).

(3) Only-if-direction: Suppose $\sigma(t) \leq \sigma(u)$. Let $i = \sigma(t) \cdot (n+1) + \sigma(u)$. Since $n \geq \max\{\sigma(t), \sigma(u)\}$, we have $i < (n+1)^2$. Furthermore, $\sigma(t) = \lfloor i/(n+1) \rfloor$ and $\sigma(u) = i \bmod (n+1)$. For $p = 4(n^2+1)^2 + 4(n+1)^2 + 3i$, we have $h_n(p) = 2$, $h_n(p+1) = \sigma(t) + 3$ by the definition of h_n. Since $\sigma(t) \leq \sigma(u)$, we have $h_n(p+2) = \sigma(u) + 3$ by the definition of h_n. From this, we have $\sigma, h_n \models \exists a(a \hookrightarrow \overline{2}, [t], [u])$, that is, $\sigma, h_n \models \mathsf{Ineq}(t, u)$.
If-direction: Suppose $\sigma, h_n \models \mathsf{Ineq}(t, u)$. Since $n \geq \max\{\sigma(t), \sigma(u)\}$, there exists p such that $h_n(p) = 2$, $h_n(p+1) = \sigma(t) + 3$, $h_n(p+2) = \sigma(u) + 3$. By Lemma 4.3 (3), we have $h_n(p+1) \leq h_n(p+2)$, that is, $\sigma(t) \leq \sigma(u)$. ◀

The next lemma shows that if Add, Mult and $\neg\mathsf{Ineq}$ are true for a sufficiently large simple table heap, they are also true for all heaps.

▶ **Lemma 4.9.** *For $n \geq \max\{\sigma(t), \sigma(u)\}$, the following hold.*

(1) *$\sigma, h_n \models \mathsf{Add}(t, u, v)$ if and only if $\sigma, h \models H \to \mathsf{Add}(t, u, v)$ for all h.*

(2) *$\sigma, h_n \models \mathsf{Mult}(t, u, v)$ if and only if $\sigma, h \models H \to \mathsf{Mult}(t, u, v)$ for all h.*

(3) *$\sigma, h_n \models \neg\mathsf{Ineq}(t, u)$ if and only if $\sigma, h \models H \to \neg\mathsf{Ineq}(t, u)$ for all h.*

Proof. The if-direction is obvious. We will show the only-if-direction.

(1) Since $\sigma, h_n \models \mathsf{Add}(t, u, v)$ by assumption, we have $\sigma(t) + \sigma(u) = \sigma(v)$ by Lemma 4.8 (1). We fix h in order to show $\sigma, h \models H \to \mathsf{Add}(t, u, v)$.

Case 1. If $\sigma, h \not\models H$, the claim follows trivially.

Case 2. Assume $\sigma, h \models H$.

Case 2.1 If $\sigma, h \models \forall a \neg(a \hookrightarrow 0, [t], [u])$, the claim follows trivially, because $\sigma, h \models \forall a((a \hookrightarrow 0, [t], [u]) \to s^3(a) \hookrightarrow [u])$.

Case 2.2 Assume $\sigma, h \models \exists a(a \hookrightarrow 0, [t], [u])$. We assume $h(p) = 0, h(p+1) = \sigma(t) + 3, h(p+2) = \sigma(u) + 3$ for arbitrary p. Since $\sigma, h \models H$, we have $h(p+3) = (h(p+1) - 3) + (h(p+2) - 3) + 3$ by Lemma 4.3 (1). Therefore, $h(p+3) = \sigma(t) + \sigma(u) + 3$. That is, $h(p+3) = \sigma(v) + 3$. Thus $\sigma, h \models s^3(a) \hookrightarrow [v]$. Then, we have $\sigma[a := p], h \models (a \hookrightarrow 0, [t], [u]) \to s^3(a) \hookrightarrow [v]$ for all p.

Hence in both cases $\sigma, h \models H \to \mathsf{Add}(t, u, v)$.

(2) The claim can be shown similarly to (1) (except it uses Lemma 4.3 (2)).

(3) By Lemma 4.8 (3), we have $\sigma \models \neg(t \leq u)$. We fix h in order to show $\sigma, h \models H \to \neg\mathsf{Ineq}(t, u)$.

Case 1. If $\sigma, h \not\models H$, the claim follows trivially.

Case 2. Assume $\sigma, h \models H$. Assume $\sigma, h \models \mathsf{Ineq}(t, u)$ for contradiction. Then, there is q such that $h(q) = 2$, $h(q+1) = \sigma(t) + 3$, $h(q+2) = \sigma(u) + 3$. By Lemma 4.3 (3), we have $\sigma(t) \leq \sigma(u)$, a contradiction. ◀

The next lemma says that the truth in PA is equivalent to the truth of the translation in SLN for a large simple table heap.

▶ **Lemma 4.10.** *For a normal formula A in PA and $n \geq \max(\sigma, A)$, $\sigma \models A$ if and only if $\sigma, h_n \models A^\circ$.*

Proof. We will show the claim by induction on A.

Case 1. A is quantifier-free. We will only show the cases for $A \equiv (t \leq u)$ since the cases $t = u$ and $t \neq u$ are obvious and the cases $A \wedge B$ and $A \vee B$ follow from the induction hypothesis. $\sigma \models t \leq u$ is equivalent to $\sigma \models \neg(u \leq t) \vee t = u$. Since $n \geq \max\{\sigma(t), \sigma(u)\}$, by Lemma 4.8 (3), $\sigma \models \neg(u \leq t)$ is equivalent to $\sigma, h_n \models \neg\mathsf{Ineq}(u, t)$. Hence, $\sigma \models t \leq u$ is equivalent to $\sigma, h_n \models \neg\mathsf{Ineq}(u, t) \vee t = u$. Since $\sigma, h_n \models H$ by Lemma 4.7, $\sigma, h_n \models \neg\mathsf{Ineq}(u, t) \vee t = u$ is equivalent to $\sigma, h_n \models H \to \neg\mathsf{Ineq}(u, t) \vee t = u$.

Case 2. $A \equiv \exists x \leq t.B$.

Only-if-direction: By assumption, there is k such that $\sigma[x := k] \models x \leq t \wedge B$. That is, $\sigma[x := k] \models x \leq t$ and $\sigma[x := k] \models B$. Thus, we have $k \leq \sigma(t)$. Since $n \geq \max(\sigma[x := k], B)$, by induction hypothesis, we have $\sigma[x := k], h_n \models B^\circ$. Furthermore, by Lemma 4.8 (3), $\sigma[x := k], h_n \models \mathsf{Ineq}(x, t)$. Thus, we have $\sigma[x := k], h_n \models \mathsf{Ineq}(x, t) \wedge B^\circ$. Hence, $\sigma[x := k], h_n \models \neg\mathsf{Ineq}(t, t) \vee (\mathsf{Ineq}(x, t) \wedge B^\circ)$.

If-direction: Suppose $\sigma[x := k], h_n \models H \to \neg\mathsf{Ineq}(t, t) \vee (\mathsf{Ineq}(x, t) \wedge B^\circ)$ for some k. Since $\sigma[x := k], h_n \models H$, we have $\sigma[x := k], h_n \models \neg\mathsf{Ineq}(t, t) \vee (\mathsf{Ineq}(x, t) \wedge B^\circ)$. Since $n \geq \max(\sigma, A) \geq \sigma(t)$, by the definition of h_n, $\sigma[x := k], h_n \models \mathsf{Ineq}(t, t)$. Thus, we have $\sigma[x := k], h_n \models \mathsf{Ineq}(x, t) \wedge B^\circ$. Since $\sigma[x := k], h_n \models \mathsf{Ineq}(x, t)$, by Lemma 4.8 (3), we have $k \leq \sigma(t)$. Then, since $k \leq \sigma(t) \leq n$, by the induction hypothesis for B, $\sigma[x := k] \models x \leq t \wedge B$.

Case 3. $A \equiv \forall x \leq t.B$. We will show the claim: for all k, $\sigma[x := k] \models \neg(x \leq t) \vee B$ if and only if $\sigma[x := k], h_n \models \neg\mathsf{Ineq}(x, t) \vee B^\circ$. If $k \leq n$, then by Lemma 4.8 (3) and the induction hypothesis for B, the claim holds. If $k > n$, then since $k > n \geq \sigma(t)$, we have $\sigma[x := k] \models \neg(x \leq t)$. On the other hand, by the definition of h_n, we have $\sigma[x := k], h_n \models \neg\mathsf{Ineq}(x, t)$. So the claim holds.

Case 4. $A \equiv \exists(x = t + u)B$. $\sigma \models \exists(x = t + u)B$ is equivalent to $\sigma[x := k] \models x = t + u$ and $\sigma[x := k] \models B$ for some k. Since $n \geq \max\{\sigma(t), \sigma(u)\}$, by Lemma 4.8 (1), $\sigma[x := k] \models x = t + u$ is equivalent to $\sigma[x := k], h_n \models \mathsf{Add}(t, u, x)$. Furthermore, since $n \geq \max(\sigma[x := k], B)$, by induction hypothesis for B, $\sigma[x := k] \models B$ is equivalent to $\sigma[x := k], h_n \models B^\circ$. Therefore, $\sigma \models A$ is equivalent to $\sigma[x := k], h_n \models \mathsf{Add}(t, u, x) \wedge B^\circ$ for some k, which is equivalent to $\sigma, h_n \models \exists x(\mathsf{Add}(t, u, x) \wedge B^\circ)$.

Case 5. $A \equiv \exists(x = y \times z)B$. This case can be shown similarly to Case 4 (except it uses Lemma 4.8 (2)). ◀

The next lemma says that for the translation of a normal formula in PA, the truth for a large simple table heap is the same as the truth for all heaps in the standard interpretation of SLN.

▶ **Lemma 4.11.** *Let A be a normal formula in* PA *and $n \geq \max(\sigma, A)$. Then, $\sigma, h_n \models A^\circ$ if and only if $\sigma, h \models A^\circ$ for all h.*

Proof. The if-direction is trivial. We will show the only-if-direction by induction on A.

Case 1. A is quantifier-free. We will only show the cases for $A \equiv (t \leq u)$ since the cases $t = u$ and $t \neq u$ are obvious and the cases $A \wedge B$ and $A \vee B$ follow from the induction hypothesis. Since $\sigma, h_n \models H$ by Lemma 4.7, $\sigma, h_n \models H \to \neg\mathsf{Ineq}(u, t) \vee t = u$ is equivalent to $\sigma, h_n \models \neg\mathsf{Ineq}(u, t) \vee t = u$. Since $n \geq \max\{\sigma(t), \sigma(u)\}$, by Lemma 4.9 (3), $\sigma, h_n \models \neg\mathsf{Ineq}(u, t)$ is equivalent to $\sigma, h \models H \to \neg\mathsf{Ineq}(u, t)$ for all h. Clearly, $\sigma, h_n \models t = u$ is equivalent to $\sigma, h \models t = u$ for all h. Therefore, we have $\sigma, h \models (H \to \neg\mathsf{Ineq}(u, t)) \vee t = u$ for all h, which is equivalent to $\sigma, h \models H \to \neg\mathsf{Ineq}(u, t) \vee t = u$ for all h.

Case 2. $A \equiv \exists x \leq t.B$. Suppose $\sigma, h_n \models H \to \neg\mathsf{lneq}(t,t) \vee \exists x(\mathsf{lneq}(x,t) \wedge B^\circ)$. Since $\sigma, h_n \models H$ and $\sigma, h_n \models \mathsf{lneq}(t,t)$, we have $\sigma, h_n \models \exists x(\mathsf{lneq}(x,t) \wedge B^\circ)$, that is, for some k, $\sigma[x := k], h_n \models \mathsf{lneq}(x,t) \wedge B^\circ$. Let this fact be (a). We fix h in order to show $\sigma, h \models H \to \neg\mathsf{lneq}(t,t) \vee \exists x(\mathsf{lneq}(x,t) \wedge B^\circ)$. Assume $\sigma, h \models H$. If $\sigma, h \models \neg\mathsf{lneq}(t,t)$, the claim trivially holds. Consider the case $\sigma, h \models \mathsf{lneq}(t,t)$. By (a), we have $\sigma[x := k], h_n \models \mathsf{lneq}(x,t)$. Thus, by Lemma 4.8 (3), $k \leq \sigma(t)$. By the case condition, $\sigma, h \models \mathsf{lneq}(t,t)$. Then, by Lemma 4.3 (4), we have $\sigma[x := k], h \models \mathsf{lneq}(x,t)$. Moreover, since $n \geq \max(\sigma[x := k], B)$, by induction hypothesis, we have $\sigma[x := k], h' \models B^\circ$ for all h'. Therefore, we have $\sigma[x := k], h \models B^\circ$. Thus, we have $\sigma[x := k], h \models \mathsf{lneq}(x,t) \wedge B^\circ$, that is, $\sigma, h \models \exists x(\mathsf{lneq}(x,t) \wedge B^\circ)$.

Case 3. $A \equiv \forall x \leq t.B$. Suppose $\sigma, h_n \models H \to \forall x(\neg\mathsf{lneq}(x,t) \vee B^\circ)$. Since $\sigma, h_n \models H$, we have $\sigma, h_n \models \forall x(\neg\mathsf{lneq}(x,t) \vee B^\circ)$. We fix h in order to show $\sigma, h \models H \to \forall x(\neg\mathsf{lneq}(x,t) \vee B^\circ)$. Assume $\sigma, h \models H$. We fix k in order to show $\sigma[x := k], h \models \neg\mathsf{lneq}(x,t) \vee B^\circ$. We consider the cases for $\sigma[x := k], h \models \mathsf{lneq}(x,t)$ and $\sigma[x := k], h \models \neg\mathsf{lneq}(x,t)$ separately.

Case 3.1. The case $\sigma[x := k], h \models \mathsf{lneq}(x,t)$. Then, there is p such that $h(p) = 2$, $h(p+1) = \sigma[x := k](x) + 3 = k + 3$, $h(p+2) = \sigma[x := k](t) + 3 = \sigma(t) + 3$. By Lemma 4.3 (3), we have $k \leq \sigma(t)$. Hence, by Lemma 4.8 (3), we have $\sigma[x := k], h_n \models \mathsf{lneq}(x,t)$. Then, $\sigma[x := k], h_n \models B^\circ$ must be the case. Since $k \leq \sigma(t) \leq n$, we apply the induction hypothesis to B and obtain $\sigma[x := k], h' \models B^\circ$ for all h'. Hence, we have $\sigma[x := k], h \models B^\circ$. Then, we have the desired result $\sigma[x := k], h \models \neg\mathsf{lneq}(x,t) \vee B^\circ$.

Case 3.2. If $\sigma[x := k], h \models \neg\mathsf{lneq}(x,t)$, then $\sigma[x := k], h \models \neg\mathsf{lneq}(x,t) \vee B^\circ$ trivially holds.

Hence in both cases, we have $\sigma[x := k], h \models \neg\mathsf{lneq}(x,t) \vee B^\circ$.

Case 4. $A \equiv \exists(x = t + u)B$. Then, $A^\circ \equiv H \to \exists x(\mathsf{Add}(t,u,x) \wedge B^\circ)$. We fix h and assume $\sigma, h \models H$ in order to show $\sigma, h \models \exists x(\mathsf{Add}(t,u,x) \wedge B^\circ)$. Since $\sigma, h_n \models H$, we have $\sigma, h_n \models \exists x(\mathsf{Add}(t,u,x) \wedge B^\circ)$. That is, there exists k such that $\sigma[x := k], h_n \models \mathsf{Add}(t,u,x) \wedge B^\circ$, which is equivalent to $\sigma[x := k], h_n \models \mathsf{Add}(t,u,x)$ and $\sigma[x := k], h_n \models B^\circ$. By Lemma 4.9 (1), $\sigma[x := k], h_n \models \mathsf{Add}(t,u,x)$ is equivalent to $\sigma[x := k], h' \models H \to \mathsf{Add}(t,u,x)$ for all h'. Since we assumed $\sigma, h \models H$, we have $\sigma[x := k], h \models \mathsf{Add}(t,u,x)$. Moreover, since $n \geq \max(\sigma[x := k], B)$, by induction hypothesis for B, we have $\sigma[x := k], h' \models B^\circ$ for all h'. Thus, we have $\sigma[x := k], h \models B^\circ$. Therefore, we have $\sigma[x := k], h \models \mathsf{Add}(t,u,x) \wedge B^\circ$, that is, $\sigma, h \models \exists x(\mathsf{Add}(t,u,x) \wedge B^\circ)$.

Case 5. $A \equiv \exists(x = y \times z)B$. This case can be shown similarly to Case 4 (except it uses Lemma 4.9 (2)). ◀

Now we have the main lemma, which says that the truth of a normal formula with $\forall$ in PA for the standard model is the same as the truth of its translation in SLN for the standard interpretation for all heaps.

▶ **Lemma 4.12.** *If A is a normal formula in* PA*, $\sigma \models \forall xA$ if and only if $\sigma, h \models (\forall xA)^\circ$ for all h.*

Proof. $\sigma \models \forall xA$ is equivalent to $\sigma[x := k] \models A$ for all $k \in \mathbb{N}$. We fix k. Let $n \geq \max(\sigma[x := k], A)$. By Lemma 4.10, $\sigma[x := k] \models A$ is equivalent to $\sigma[x := k], h_n \models A^\circ$. Then, by Lemma 4.11, this is equivalent to $\sigma[x := k], h \models A^\circ$ for all h. Therefore, $\sigma[x := k] \models A$ is equivalent to $\sigma[x := k], h \models A^\circ$ for all h. Hence, $\sigma[x := k] \models A$ for all k is equivalent to $\sigma[x := k], h \models A^\circ$ for all h for all k. Thus, $\sigma \models \forall xA$ is equivalent to $\sigma, h \models \forall xA^\circ$ for all h, that is, $\sigma, h \models (\forall xA)^\circ$ for all h. ◀

5 Translation from PA into SLN

In this section, we will present the translation of a Π^0_1 formula in PA to a formula in SLN and prove that the translation preserves the validity and the non-validity. In order to define the translation, first we will define a translation of a Π^0_1 formula in PA into an equivalent normal formula with one universal quantifier in PA. Finally we will define the translation by combining the two translations and will present the main theorem, which says a Π^0_1 formula in PA can be simulated in the weak fragment SLN of separation logic. We also discuss a counterexample for the translation when we extend it to Σ^0_1 formulas.

First we will transform a Π^0_1 formula in PA into a normal formula with one universal quantifier in PA. For simplicity, we use vector notation $\overrightarrow{e}$ for a sequence $e_1, ..., e_n$ of objects.

▶ **Proposition 5.1.** *If A is a bounded formula in* PA*, there is a normal formula B such that $A \leftrightarrow B$ is valid.*

Proof. First, transform A into a prenex normal form and replace $\neg(t \leq u)$ by $u \leq t \wedge u \neq t$ to obtain $A' \equiv \overrightarrow{Qx \leq t}.C$, where C is a quantifier-free disjunctive normal form without formulas of the form $\neg(t \leq u)$. Choose the leftmost occurrence among the innermost occurrences of $u + v$ or $u \times v$ in A' and explicitly denote it by $A'[u + v]$ or $A'[u \times v]$.

Let $A'[z]$ be the formula obtained from $A'[u + v]$ or $A'[u \times v]$ by replacing the occurrence of $u + v$ or $u \times v$ in A' by a fresh z. Define $\overrightarrow{Qx' \leq t'}.D$ by $A'[z] \equiv \overrightarrow{Qx' \leq t'}.D$ where $\overrightarrow{Qx' \leq t'}$ is the longest prefix such that z is not in t', namely, it has the longest $\overrightarrow{Qx' \leq t'}$ among such $\overrightarrow{Qx' \leq t'}$'s. We transform D into $\exists(z = u + v)D$ or $\exists(z = u \times v)D$.

We repeat this process until we have the form $\overrightarrow{\{Qx \leq y, \exists(x = t)\}}A''$, where t is of the form $a + b$ or $a \times b$ for some terms a, b that do not contain $+$ or $\times$, and A'' and u do not contain $+, \times$. Define B as this result. ◀

We define the translation $A^\square$ by using the proof of the previous proposition.

▶ **Definition 5.2** (Translation $(\cdot)^\square$)**.** Let $A \equiv \forall x B$ be a Π^0_1 formula in PA, where B contains only bounded quantifiers. Let B' be a normal form of B obtained by the procedure described in the proof of Proposition 5.1. We define $A^\square \equiv \forall x B'$.

Example. For a formula

$$A \equiv \forall y \leq x + (x + s(x)).(0 \leq x + (y \times (x + y))),$$

its translation $A^\square$ is

$$\begin{aligned} A^\square \equiv & \exists(x_1 = x + s(x))\exists(x_2 = x + x_1)\forall y \leq x_2. \\ & \exists(x_3 = x + y)\exists(x_4 = y \times x_3)\exists(x_5 = x + x_4)(0 \leq x_5). \end{aligned}$$

Now, we have the main theorem which says that Π^0_1 formulas can be translated into SLN formulas preserving the validity and the non-validity.

▶ **Theorem 5.3.** *For a Π^0_1 formula A in* PA*, A is valid in the standard model of* PA *if and only if $A^{\square\circ}$ is valid in the standard interpretation of* SLN*.*

Proof. By Proposition 5.1 and Lemma 4.12. ◀

As a by-product of the above theorem, we have the undecidability of SLN.

▶ **Corollary 5.4.** *The validity of* SLN *formulas is undecidable.*

Proof. Given a Turing machine, its halting problem statement P is Σ^0_1, since it can be expressed as $\exists z.T(e,e,z)$, where e is the index of the given Turing machine and T is Kleene's T-predicate which is primitive recursive (for rigorous definition, see e.g. [13]). Thus, $\neg P$ is Π^0_1. By Theorem 5.3, $\neg P$ is valid in PA if and only if $(\neg P)^{\Box\circ}$ is valid in SLN. If the validity in SLN is decidable, we can decide whether P is true in the standard model and this contradicts the undecidability of the halting problem. Hence the validity in SLN is undecidable. ◀

We have just proved that Π^0_1 formulas can be translated preserving the validity and the non-validity. We might extend the translation $(\cdot)^\circ$ by $(\exists x A)^\circ \equiv \exists x A^\circ$. However, the extended translation does not preserve the validity and the non-validity, which is shown in the next proposition.

▶ **Proposition 5.5.** *There is some Σ^0_1 closed formula A such that A is not valid in* PA *but $A^{\Box\circ}$ is valid in* SLN.

Proof. Consider the formula $A \equiv \exists x(x+0 \neq x)$. This sentence is clearly not valid in PA. However, we can prove that $\sigma, h \models A^{\Box\circ}$ for all σ, h as follows. By the procedure in the proof of Proposition 5.1, $A^{\Box} \equiv \exists x \exists (z = x+0)(z \neq x)$. Thus, $A^{\Box\circ} \equiv \exists x(H \to \exists z(\mathsf{Add}(x,0,z) \land z \neq x))$. We fix σ, h in order to prove $\sigma, h \models A^{\Box\circ}$. Let $n = \max\{k \mid h(p) = 0, h(p+1) = k+3, h(p+2) = 3\} + 1$, and $m = n+1$. Let $\sigma' = \sigma[x := n, z := m]$. We will show $\sigma', h \models \mathsf{Add}(x,0,z) \land z \neq x$ assuming $\sigma', h \models H$. By choice of n, we have $\sigma', h \models \forall a \neg(a \hookrightarrow 0, s^3(\overline{n}), \overline{3})$. Thus, $\sigma', h \models \mathsf{Add}(\overline{n}, 0, \overline{m})$ holds, because the premise of $\mathsf{Add}(\overline{n}, 0, \overline{m})$ is false. Therefore, $\sigma', h \models \mathsf{Add}(x,0,z)$. Furthermore, clearly $\sigma', h \models z \neq x$. Hence, $\sigma, h \models A^{\Box\circ}$ for all h. ◀

6 Another undecidability proof

In this section, we will give another proof of the undecidability of the validity of SLN given in Corollary 5.4, where it can be proved in a way similar to that in [7]. This proof is simpler than the proof of Theorem 5.3, but this proof cannot show the representation of Peano arithmetic in the separation logic SLN with numbers.

A first-order language L is defined as that with a binary predicate symbol P and without any constants or function symbols. Namely,

Terms $t ::= x$.

Formulas $A ::= t = t \mid P(t,t) \mid \neg A \mid A \land A \mid \exists x.A$.

A finite structure is defined as (U, R) where $U \subseteq \mathbb{N}$ and U is finite and $R \subseteq U^2$. σ is a variable assignment of (U, R) if $\sigma : \mathrm{Vars} \to U$. We define σ_0 as $\sigma_0(x) = 0$ for all variables x.

We write $M, \sigma \models A$ to denote that a formula A is true by a variable assignment σ of a structure M.

The idea of this proof is to encode a finite structure (U, R) for the language L by a heap h such that

- $n \in U$ iff h has some entry of $0, n+2$, and
- $(n,m) \in R$ iff h has some entry of $1, n+2, m+2$.

▶ **Definition 6.1.** For a given finite structure $M = (U, R)$ of L, we define the heap h_M by

$$\begin{aligned}
&\mathsf{Dom}(h) = \{0, 1, \ldots, 2k + 3l - 1\},\\
&h_M(x) = 0 \quad (x = 2i, i < k),\\
&h_M(x) = p_i + 2 \quad (x = 2i + 1, i < k),\\
&h_M(x) = 1 \quad (x = 2k + 3i, i < l),\\
&h_M(x) = n_i + 2 \quad (x = 2k + 3i + 1, i < l),\\
&h_M(x) = m_i + 2 \quad (x = 2k + 3i + 2, i < l),
\end{aligned}$$

where $U = \{p_i \mid i < k\}$ and $R = \{(n_i, m_i) \mid i < l\}$.

The heap h_M has information of a given structure M.

▶ **Definition 6.2.** For a given heap h, if $\sigma_0, h \models \exists a x(a \hookrightarrow 0, s^2(x))$, we define a structure $M_h = (U_h, R_h)$ by

$$\begin{aligned}
&U_h = \{n \mid \sigma_0[x := n], h \models \exists a(a \hookrightarrow 0, s^2(x))\},\\
&R_h = \{(n, m) \mid \sigma_0[x := n, y := m], h \models \exists a(a \hookrightarrow \overline{1}, s^2(x), s^2(y))\}.
\end{aligned}$$

The structure M_h is a structure represented by a given heap h.

We define a translation $(\cdot)^\triangle$ from L into SLN.

▶ **Definition 6.3.** For a formula A in the language L, we define the formula $A^\triangle$ in SLN by

$$\begin{aligned}
&(x = y)^\triangle \equiv x = y \wedge \exists a(a \hookrightarrow 0, s^2(x)),\\
&(P(x, y))^\triangle \equiv \exists a(a \hookrightarrow \overline{1}, s^2(x), s^2(y)) \wedge \exists b(b \hookrightarrow 0, s^2(x)) \wedge \exists c(c \hookrightarrow 0, s^2(y)),\\
&(\exists x.A)^\triangle \equiv \exists x(\exists a(a \hookrightarrow 0, s^2(x)) \wedge A^\triangle),\\
&(\neg A)^\triangle \equiv \neg A^\triangle,\\
&(A \wedge B)^\triangle \equiv A^\triangle \wedge B^\triangle.
\end{aligned}$$

The next is a well-known theorem for finite structures [9].

▶ **Theorem 6.4** (Trakhtenbrot). *The validity of formulas in the language L for every finite structure is undecidable.*

The next lemma shows the equivalence for any formulas.

▶ **Lemma 6.5.** *$M, \sigma \models A$ for all finite M for all variable assignments σ of M iff $\sigma, h \models \exists a x(a \hookrightarrow 0, s^2(x)) \to \bigwedge_{x \in \mathsf{FV}(A)} \exists a(a \hookrightarrow 0, s^2(x)) \to A^\triangle$ for all h and all variable assignments σ.*

Proof. If-direction: For a given finite structure M, we can construct the heap h_M and by induction on A we can show that $\sigma, h_M \models A^\triangle$ iff $M, \sigma \models A$, for every variable assignment σ of M.

Only-if-direction: For a given heap h such that $\sigma_0, h \models \exists a x(a \hookrightarrow 0, s^2(x))$, we can construct the finite structure M_h and by induction on A we can show that $M_h, \sigma \models A$ iff $\sigma, h \models A^\triangle$, for every variable assignment σ of M_h. To show the only-if-direction in the statement of the lemma by using this claim, from the assumption $\sigma_0, h \models \exists a x(a \hookrightarrow 0, s^2(x))$, we have p, q such that $h(p) = 0$ and $h(p + 1) = q + 2$, and for a given σ we apply this claim with the variable assignment σ' of M_h such that $\sigma'(x) = \sigma(x)$ $(x \in \mathsf{FV}(A))$ and $\sigma'(x) = q$ $(otherwise)$. ◀

Another proof of Corollary 5.4. Taking a closed formula A in Lemma 6.5, we have the equivalence: A is true in all finite structure M iff $\exists ax(a \hookrightarrow 0, s^2(x)) \to A^{\triangle}$ is valid in the standard interpretation of SLN.

By Theorem 6.4, the validity of SLN for the standard interpretation is undecidable. ◀

7 Conclusion

We have presented the translation from Π_1^0 formulas in PA into formulas in the fragment SLN of separation logic with numbers, which has only $\hookrightarrow$, 0 and the successor function, and proved that this translation preserves the validity and the non-validity for the standard model of PA and the standard interpretation of SLN. By this, we have shown that Π_1^0 formulas in Peano arithmetic can be simulated by SLN. As a corollary, we have proved the validity of SLN is undecidable. We have also given a counterexample when we extend this translation to Σ_1^0 formulas.

Future work would be to present a translation from other logical systems into SLN, and prove the preservation of the validity and the non-validity by extending our technique used in this paper. Another future work would be to try to show the validity of SLN is Π_1^0-complete.

References

1 Timos Antonopoulos, Nikos Gorogiannis, Christoph Haase, Max I. Kanovich, and Joël Ouaknine. Foundations for decision problems in separation logic with general inductive predicates. In Anca Muscholl, editor, *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*, volume 8412 of *Lecture Notes in Computer Science*, pages 411–425. Springer, 2014. doi:10.1007/978-3-642-54830-7_27.

2 Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. A decidable fragment of separation logic. In Kamal Lodaya and Meena Mahajan, editors, *FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science, 24th International Conference, Chennai, India, December 16-18, 2004, Proceedings*, volume 3328 of *Lecture Notes in Computer Science*, pages 97–109. Springer, 2004. doi:10.1007/978-3-540-30538-5_9.

3 Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. Symbolic execution with separation logic. In Kwangkeun Yi, editor, *Programming Languages and Systems, Third Asian Symposium, APLAS 2005, Tsukuba, Japan, November 2-5, 2005, Proceedings*, volume 3780 of *Lecture Notes in Computer Science*, pages 52–68. Springer, 2005. doi:10.1007/11575467_5.

4 Rémi Brochenin, Stéphane Demri, and Étienne Lozes. On the almighty wand. *Inf. Comput.*, 211:106–137, 2012. doi:10.1016/j.ic.2011.12.003.

5 Cristiano Calcagno, Dino Distefano, Peter W. O'Hearn, and Hongseok Yang. Compositional shape analysis by means of bi-abduction. *J. ACM*, 58(6):26:1–26:66, 2011. doi:10.1145/2049697.2049700.

6 Cristiano Calcagno, Philippa Gardner, and Matthew Hague. From separation logic to first-order logic. In Vladimiro Sassone, editor, *Foundations of Software Science and Computational Structures, 8th International Conference, FOSSACS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, volume 3441 of *Lecture Notes in Computer Science*, pages 395–409. Springer, 2005. doi:10.1007/978-3-540-31982-5_25.

7 Cristiano Calcagno, Hongseok Yang, and Peter W. O'Hearn. Computability and complexity results for a spatial assertion language for data structures. In *The Second Asian Workshop on Programming Languages and Systems, APLAS'01, Korea Advanced Institute of Science and Technology, Daejeon, Korea, December 17-18, 2001, Proceedings*, pages 289–300, 2001.

8 Byron Cook, Christoph Haase, Joël Ouaknine, Matthew J. Parkinson, and James Worrell. Tractable reasoning in a fragment of separation logic. In Joost-Pieter Katoen and Barbara König, editors, *CONCUR 2011 – Concurrency Theory – 22nd International Conference, CONCUR 2011, Aachen, Germany, September 6-9, 2011. Proceedings*, volume 6901 of *Lecture Notes in Computer Science*, pages 235–249. Springer, 2011. doi:10.1007/978-3-642-23217-6_16.

9 Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite Model Theory*. Springer-Verlag, 1995.

10 Radu Iosif, Adam Rogalewicz, and Jirí Simácek. The tree width of separation logic with recursive definitions. In Maria Paola Bonacina, editor, *Automated Deduction – CADE-24 – 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings*, volume 7898 of *Lecture Notes in Computer Science*, pages 21–38. Springer, 2013. doi:10.1007/978-3-642-38574-2_2.

11 Koji Nakazawa, Makoto Tatsuta, Daisuke Kimura, and Mitsuru Yamamura. Spatial factorization in cyclic-proof system for separation logic. *Computer Software*, 37(1):1_125–1_144, 2020. doi:10.11309/jssst.37.1_125.

12 Peter W. O'Hearn. Separation logic. *Commun. ACM*, 62(2):86–95, 2019. doi:10.1145/3211968.

13 Joseph R. Shoenfield. *Mathematical Logic*. Addison-Wesley, 1967.

14 Makoto Tatsuta and Daisuke Kimura. Separation logic with monadic inductive definitions and implicit existentials. In Xinyu Feng and Sungwoo Park, editors, *Programming Languages and Systems – 13th Asian Symposium, APLAS 2015, Pohang, South Korea, November 30 – December 2, 2015, Proceedings*, volume 9458 of *Lecture Notes in Computer Science*, pages 69–89. Springer, 2015. doi:10.1007/978-3-319-26529-2_5.

15 Makoto Tatsuta, Quang Loc Le, and Wei-Ngan Chin. Decision procedure for separation logic with inductive definitions and presburger arithmetic. In Atsushi Igarashi, editor, *Programming Languages and Systems – 14th Asian Symposium, APLAS 2016, Hanoi, Vietnam, November 21-23, 2016, Proceedings*, volume 10017 of *Lecture Notes in Computer Science*, pages 423–443, 2016. doi:10.1007/978-3-319-47958-3_22.

16 Makoto Tatsuta, Koji Nakazawa, and Daisuke Kimura. Completeness of cyclic proofs for symbolic heaps with inductive definitions. In Anthony Widjaja Lin, editor, *Programming Languages and Systems – 17th Asian Symposium, APLAS 2019, Nusa Dua, Bali, Indonesia, December 1-4, 2019, Proceedings*, volume 11893 of *Lecture Notes in Computer Science*, pages 367–387. Springer, 2019. doi:10.1007/978-3-030-34175-6_19.

Semantics for a Turing-Complete Reversible Programming Language with Inductive Types

Kostia Chardonnet
Department of Computer Science and Engineering, University of Bologna, Italy

Louis Lemonnier[1]
Université Paris-Saclay, CNRS, ENS Paris-Saclay, Inria, Laboratoire Méthodes Formelles, Gif-sur-Yvette, France

Benoît Valiron
Université Paris-Saclay, CNRS, CentraleSupélec, ENS Paris-Saclay, Inria, Laboratoire Méthodes Formelles, Gif-sur-Yvette, France

Abstract

This paper is concerned with the expressivity and denotational semantics of a functional higher-order reversible programming language based on Theseus. In this language, pattern-matching is used to ensure the reversibility of functions. We show how one can encode any Reversible Turing Machine in said language. We then build a sound and adequate categorical semantics based on join inverse categories, with additional structures to capture pattern-matching and to interpret inductive types and recursion. We then derive a notion of completeness in the sense that any computable, partial, first-order injective function is the image of a term in the language.

2012 ACM Subject Classification Theory of computation → Program semantics

Keywords and phrases Reversible programming, functional programming, Computability, Denotational Semantics

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.19

Funding This work has been partially funded by the French National Research Agency (ANR) by the projects ANR-21-CE48-0019, ANR-19-CE48-0014, ANR-22-CE47-0012, and within the framework of "Plan France 2030", under the project ANR-22-PETQ-0007, ANR-22-PNCQ-0001.
Kostia Chardonnet: is partially supported by the MIUR FARE project CAFFEINE, "Compositional and Effectful Program Distances", R20LW7EJ7L.

Acknowledgements The authors thank Vladimir Zamdzhiev for his expert insight on specific parts of the denotational semantics.

1 Introduction

Originally, reversible computation has emerged as an energy-preserving model of computation in which no data is ever erased. This comes from Laundauer's principle which states that the erasure of information is linked to the dissipation of energy as heat [30, 5]. In reversible computation, given some process f, there always exists an inverse process f^{-1} such that their composition is equal to the identity: it is always possible to "*go back in time*" and recover the input of your computation. Although this can be seen as very restrictive, non-reversible computation can be emulated in a reversible setting by keeping track of intermediate results. As discussed in [4], the simulation of standard computation with reversible computation can be understood as a notion of *Turing completeness* – provided we accept that the final result comes together with auxiliary, intermediate computation.

[1] Corresponding author.

Reversible computation has since been shown to be a versatile model. In the realm of quantum computation, reversible computing is at the root of the construction of *oracles*, subroutines describing problem instances in quantum algorithms [34]. Most of the research in reversible circuit design can then been repurposed to design efficient quantum circuits. On the theoretical side, reversible computing serves the main ingredient in several operational models of linear logic, whether through token-based Geometry of Interaction [32] or through the Curry-Howard correspondence for μMALL [8, 6].

Reversible programming has been approached in two different ways. The first one, based on Janus and later R-CORE and R-WHILE [31, 40, 15, 39], considers imperative and flow-chart languages. The other one follows a functional approach [38, 37, 20, 19, 36, 8]: a function $A \to B$ in the language represents a function – a bijection – between values of type A and values of type B. In this approach, types are typically structured, and functional reversible languages usually feature pattern-matching to discriminate on values.

One of the issues reversible programming has to deal with is *non-termination*: in general, a reversible program computes a *partial injective map*. This intuition can be formalised with the concept of *inverse categories* [27, 9, 10, 11]: categories in which every morphism comes with a partial inverse, for which the category **PInj** of sets and partial injective maps is the emblematic concrete instance. This categorical setting has been successfully used in the study of reversible programming semantics, whether based on flow-charts [14, 22], with recursion [2, 24, 23, 26], with side effects [18, 17], *etc.*

Although much work has been dedicated to the categorical analysis of reversible computation, the *adequacy* of the developed categorical constructs with reversible functional programming languages has only recently been under scrutiny, either in *concrete* categories of partial isomorphisms [26, 25], or for simple, *non Turing-complete* languages [7]. A formal, categorical analysis of a (reversible) Turing-complete, reversible language is still missing.

Contributions. In this paper, we aim at closing this gap: we propose a Turing-complete (understood as in the reversible setting), reversible language, together with a categorical semantics. In particular, the contributions of this paper are as follows.

- A (reversible) Turing-complete, higher-order reversible language with inductive types. Building on the Theseus-based family of languages studied in [36, 7, 8, 6], we consider an extension with *inductive types*, general *recursion* and *higher-order* functions.
- Sound and adequate categorical semantics. We show how the language can be interpreted in join inverse rig categories. The result relies on the **DCPO**-enrichments of join inverse rig categories.
- A notion of completeness. We finally discuss how the interpretation of the language in the category **PInj** is complete in the sense that any *first-order* computable, partial injective function on the images of types is realisable within the language.

2 Language

In this section, we present a reversible language, unifying and extending the Theseus-based variants presented in the literature [36, 7, 8]. In particular, the language we propose features higher-order (unlike [7]), pairing, injection, inductive types (unlike [36]) and general recursion (unlike [8]). Functions in the language are based on pattern-matching, following a strict syntactic discipline: term variables in patterns should be used linearly, and clauses should be non-overlapping on the left *and* on the right (therefore enforcing non-ambiguity and injectivity). In [36, 7, 8] one also requires exhaustivity for totality. In this paper, we drop this condition in order to allow non-terminating behaviour.

Table 1 Grammar for terms and types.

$$
\begin{array}{lll}
\text{(Base types)} & A, B ::= \mathbb{1} \mid A \oplus B \mid A \otimes B \mid \mu X.A \mid X \\
\text{(Isos)} & T ::= A \leftrightarrow B \mid T_1 \to T_2 \\
\\
\text{(Values)} & v ::= () \mid x \mid \mathtt{inj}_\ell\, v \mid \mathtt{inj}_r\, v \mid \langle v_1, v_2 \rangle \mid \mathtt{fold}\, v \\
\text{(Patterns)} & p ::= x \mid \langle p_1, p_2 \rangle \\
\text{(Expressions)} & e ::= v \mid \mathtt{let}\, p_1 = \omega\ p_2\ \mathtt{in}\ e \\
\text{(Isos)} & \omega ::= \{v_1 \leftrightarrow e_1 \mid \cdots \mid v_n \leftrightarrow e_n\} \mid \mathtt{fix}\ \phi.\omega \mid \lambda\psi.\omega \mid \phi \mid \omega_1\ \omega_2 \\
\text{(Terms)} & t ::= () \mid x \mid \mathtt{inj}_\ell\, t \mid \mathtt{inj}_r\, t \mid \langle t_1, t_2 \rangle \mid \\
& \qquad \mathtt{fold}\, t \mid \omega\ t \mid \mathtt{let}\, p = t_1\ \mathtt{in}\ t_2
\end{array}
$$

The language is presented in Table 1. It consist of two layers.

- Base types: The base types consist of the unit type $\mathbb{1}$ along with its sole constructor $()$, coproduct $A \oplus B$ and tensor product $A \otimes B$ with their respective constructors, $\mathtt{inj}_\ell\ (t), \mathtt{inj}_r\ (t)$ and $\langle t_1, t_2 \rangle$. Finally, the language features inductive types of the form $\mu X.A$ where X is a type variable occurring in A and μ is its binder. Its associated constructor is $\mathtt{fold}\ (t)$. The inductive type $\mu X.A$ can then be unfolded into $A[\mu X.A/X]$, i.e., substituting each occurrence of X by $\mu X.A$ in A. Typical examples of inductive types that can be encoded this way are the natural number, as $\mathtt{nat} = \mu X.(\mathbb{1} \oplus X)$ or the lists of types A, noted $[A] = \mu X.\mathbb{1} \oplus (A \otimes X)$. Note that we only work with closed types. We shall denote term-variables with x, y, z.
- Isos types: The language features isos, denoted ω, higher order reversible functions whose types T consist either of a pair of base type, noted $A \leftrightarrow B$ or function types between isos, $T_1 \to T_2$. Note that the word *iso* comes from isomorphism. However, in this paper, we have freed some constraints; in our case, isos are *forward deterministic* and *backward deterministic*, meaning that each value has at most one image and at most one value that has the former as image. A first-order iso of type $A \leftrightarrow B$ consists of a finite set of *clauses*, written $v \leftrightarrow e$ where v is a value of type A and e an expression of type B. An expression consists of a succession of applications of isos to some argument, described by $\mathtt{let}$ constructions: $\mathtt{let}\,(x_1, \ldots, x_n) = \omega\ (y_1, \ldots, y_n)\ \mathtt{in}\ e$. Isos can take other isos as arguments through the $\lambda\phi.\omega$ construction. Finally, isos can also represent *recursive computation* through the $\mathtt{fix}\ \phi.\omega$ construction, where ϕ is an *iso-variable*. In general, we shall denote iso-variable by $\phi_1, \phi_2, \ldots$ and we use the shorthands $\mathtt{fix}\ \overrightarrow{\phi}$ or $\mathtt{fix}\ \phi_1, \ldots, \phi_n$ and $\lambda\overrightarrow{\phi}$ or $\lambda\phi_1, \ldots, \phi_n$ for $\mathtt{fix}\ \phi_1.\mathtt{fix}\ \phi_2. \ldots .\mathtt{fix}\ \phi_n.$ and $\lambda\phi.\lambda\phi_2. \ldots .\lambda\phi_n$.

Convention. We write $(t_1, \ldots, t_n)$ for $\langle t_1, \langle \ldots, t_n \rangle \rangle$ and $\bigoplus^n A$ (resp. $\bigotimes^n A$) for $A \oplus \cdots \oplus A$ (resp. $A \otimes \cdots \otimes A$) n times and $\omega_1 \ldots \omega_n t$ for a succession of $\mathtt{let}$ constructions applying ω_n to ω_1. We also consider constructors to be right-associative, meaning that $\mathtt{fold}\ \mathtt{inj}_r\ \langle x, y \rangle$ should be read as $\mathtt{fold}\ (\mathtt{inj}_r\ (\langle x, y \rangle))$. To avoid conflicts between variables, we will always work up to α-conversion and use Barendregt's convention [3, p.26], which consists of keeping the names of all bound and free variables distinct, even when this remains implicit.

Typing judgements. Both base terms and isos feature their typing judgements, given in Table 2 and Table 3. Term typing judgements are of the form $\Psi; \Delta \vdash t : A$ where Δ is a context of term-variables of type A and Ψ is a context of iso-variables of type T and isos

Table 2 Typing rules for terms.

$$\frac{}{\Psi;\emptyset \vdash () : \mathbb{1}} \qquad \frac{}{\Psi; x : A \vdash x : A} \qquad \frac{\Psi;\Delta \vdash t : A}{\Psi;\Delta \vdash \mathtt{inj}_\ell\, t : A \oplus B} \qquad \frac{\Psi;\Delta \vdash t : B}{\Psi;\Delta \vdash \mathtt{inj}_r\, t : A \oplus B}$$

$$\frac{\Psi;\Delta_1 \vdash t_1 : A \quad \Psi;\Delta_2 \vdash t_2 : B}{\Psi;\Delta_1,\Delta_2; \vdash \langle t_1, t_2\rangle : A \otimes B} \qquad \frac{\Psi;\Delta \vdash t : A[\mu X.A/X]}{\Psi;\Delta \vdash \mathtt{fold}\, t : \mu X.A} \qquad \frac{\Psi \vdash_\omega \omega : A \leftrightarrow B \quad \Psi;\Delta \vdash t : A}{\Psi;\Delta \vdash \omega\ t : B}$$

$$\frac{\Psi;\Delta_1 \vdash t_1 : A_1 \otimes \cdots \otimes A_n \qquad \Psi;\Delta_2, x_1 : A_1, \ldots, x_n : A_n \vdash t_2 : B}{\Psi;\Delta_1,\Delta_2 \vdash \mathtt{let}\,(x_1,\ldots,x_n) = t_1\ \mathtt{in}\ t_2 : B}$$

Table 3 Typing rules for isos.

$$\frac{}{\Psi,\phi : T \vdash_\omega \phi : T} \qquad \frac{\Psi,\phi : T \vdash_\omega \omega : T}{\Psi \vdash_\omega \mathtt{fix}\ \phi.\omega : T}$$

$$\frac{\Psi \vdash_\omega \omega_1 : T_1 \quad \Psi \vdash_\omega \omega_2 : T_1 \to T_2}{\Psi \vdash_\omega \omega_2\ \omega_1 : T_2} \qquad \frac{\Psi,\phi : T_1 \vdash_\omega \omega : T_2}{\Psi \vdash_\omega \lambda\phi.\omega : T_1 \to T_2}$$

$$\frac{\begin{array}{llll}\Psi;\Delta_1 \vdash v_1 : A & \ldots & \Psi;\Delta_n \vdash v_n : A & \forall i \neq j, v_i \perp v_j \\ \Psi;\Delta_1 \vdash e_1 : B & \ldots & \Psi;\Delta_n \vdash e_n : B & \forall i \neq j, e_i \perp e_j\end{array}}{\Psi \vdash_\omega \{v_1 \leftrightarrow e_1 \mid \cdots \mid v_n \leftrightarrow e_n\} : A \leftrightarrow B.}$$

typing judgements are of the form $\Psi \vdash_\omega \omega : T$. While Δ is a *linear* context, Ψ is not, as an iso represents a closed computation, and can be duplicated or erased at will. In the last rule of Table 3, the term variables in Δ are bound by the pattern-matching construction: they are not visible outside of the term, thus not appearing anymore in the typing context of the conclusion.

While [8] and [36] require isos to be exhaustive (i.e. to cover all the possible values of their input types) and non-overlapping (i.e. two clauses cannot match the same value), we relax the exhaustivity requirement in this paper, in the spirit of what was done in [7]. Non-overlapping is formalised by the notion of *orthogonality* between values, noted $v_1 \perp v_2$.

▶ **Definition 1** (Orthogonality). *We introduce a binary relation $\perp$ on terms. Given two terms t_1, t_2, $t_1 \perp t_2$ holds if it can be derived inductively with the rules below; we say that t_1 and t_2 are orthogonal. The relation $\perp$ is defined as the smallest relation such that:*

$$\frac{}{\mathtt{inj}_\ell\, t_1 \perp \mathtt{inj}_r\, t_2} \qquad \frac{}{\mathtt{inj}_r\, t_1 \perp \mathtt{inj}_\ell\, t_2} \qquad \frac{t_1 \perp t_2}{C_\perp[t_1] \perp C_\perp[t_2],}$$

where the contexts $C_\perp$ are defined using the following grammar:

$$C_\perp ::= [-] \mid \mathtt{inj}_\ell\, C_\perp \mid \mathtt{inj}_r\, C_\perp \mid \langle C_\perp, t\rangle \mid \langle t, C_\perp\rangle \mid \mathtt{fold}\, C_\perp \mid \mathtt{let}\, p = t\ \mathtt{in}\ C_\perp$$

Operational semantics. The language comes equipped with a rewriting system $\to$ on terms, defined in Table 4. As usual, we write $\to^*$ for the reflexive transitive closure of $\to$. The evaluation contexts $C_\to$ are defined by the grammar $[\,] \mid \mathtt{inj}_\ell\, C_\to \mid \mathtt{inj}_r\, C_\to \mid \{v_1 \leftrightarrow e_1 \mid \cdots \mid v_n \leftrightarrow e_n\}\, C_\to \mid \mathtt{let}\, p = C_\to\ \mathtt{in}\ t \mid \langle C_\to, v\rangle \mid \langle v, C_\to\rangle \mid C_\to\ t \mid \mathtt{fold}\, C_\to$. Note how the rewriting system follows a *call-by-value* strategy on terms and values, requiring that

Table 4 Evaluation relation $\to$.

$$\frac{}{\texttt{fix}\ \phi.\omega \to \omega[\texttt{fix}\ \phi.\omega/\phi]} \qquad \frac{}{(\lambda\phi.\omega_1)\omega_2 \to \omega_1[\omega_2/\phi]} \qquad \frac{\omega_1 \to \omega_1'}{\omega_1\omega_2 \to \omega_1'\omega_2} \qquad \frac{\omega \to \omega'}{\omega\ t \to \omega'\ t}$$

$$\frac{\sigma(v_i) = v'}{\{v_1 \leftrightarrow e_1 \mid \cdots \mid v_n \leftrightarrow e_n\}\ v' \to \sigma(e_i)} \qquad \frac{t_1 \to t_2}{C_\to[t_1] \to C_\to[t_2]} \qquad \frac{\sigma(p) = v}{\texttt{let}\ p = v\ \texttt{in}\ t \to \sigma(t)}$$

the argument of an iso be fully evaluated to a value before firing the substitution. On the contrary, we follow a *call-by-name* strategy to simplify the manipulation of the fixpoint. Note that unlike [8, 36], we do not require any form of termination and isos are not required to be exhaustive: the rewriting system can diverge or be stuck. The evaluation of an iso applied to a value is dealt with by pattern-matching: the input value will try to match one of the values from the clauses and potentially create a substitution if the two values match, giving the corresponding expression as an output under that substitution. A substitution σ is a mapping from a set of variables to terms. The substitution of σ on an expression t, written $\sigma(t)$, is defined in the usual way by $\sigma(()) = ()$; $\sigma(x) = v$ if $\{x \mapsto v\} \subseteq \sigma$; $\sigma(\texttt{inj}_r\ (t)) = \texttt{inj}_r\ (\sigma(t))$; $\sigma(\texttt{inj}_\ell\ (t)) = \texttt{inj}_\ell\ (\sigma(t))$; $\sigma(\texttt{fold}\ (t)) = \texttt{fold}\ (\sigma(t))$ $\sigma(\langle t, t'\rangle) = \langle\sigma(t), \sigma(t')\rangle$; $\sigma(\omega\ t) = \omega\ \sigma(t)$ and $\sigma(\texttt{let}\ p = t_1\ \texttt{in}\ t_2) = (\texttt{let}\ p = \sigma(t_1)\ \texttt{in}\ \sigma(t_2))$. The support of a substitution, written $\text{supp}(\sigma)$, is defined as $\{x \mid (x \mapsto v) \in \sigma\}$.

▶ **Lemma 2** (Subject Reduction). *If $\Psi; \Delta \vdash t : A$ and $t \to t'$, then $\Psi; \Delta \vdash t' : A$.* ⌟

The proof is similar to what has been done in [8]. As the rewriting system is deterministic, confluence is direct; meanwhile, as we are concerned with partial functions, progress is not guaranteed: a term can be stuck, for example, $\{\texttt{inj}_\ell\ (x) \leftrightarrow e\}\ \texttt{inj}_r\ (v)$ does not reduce.

Inversion. Finally, any iso $\omega : T$ can be inverted into an iso $\omega^{-1} : T^{-1}$, such that their composition makes up the identity. Intuitively, if ω is of type $A \leftrightarrow B$, then ω^{-1} will be of type $B \leftrightarrow A$. Inversion is defined as follows. Given an iso-type T, we define its inverse T^{-1} as: $(A \leftrightarrow B)^{-1} = B \leftrightarrow A$ and $(T_1 \to T_2)^{-1} = T_1^{-1} \to T_2^{-1}$. Given an iso ω, we define its dual ω^{-1} as: $\phi^{-1} = \phi$; $(\texttt{fix}\ \phi.\omega)^{-1} = \texttt{fix}\ \phi.\omega^{-1}$; $(\omega_1\ \omega_2)^{-1} = (\omega_1)^{-1}(\omega_2)^{-1}$; $(\lambda\phi.\omega)^{-1} = \lambda\phi.(\omega)^{-1}$ and $\{(v_i \leftrightarrow e_i)_{i\in I}\}^{-1} = \{((v_i \leftrightarrow e_i)^{-1})_{i\in I}\}$ and

$$\begin{pmatrix} v_1 & \leftrightarrow & \texttt{let}\ p_1 = \omega_1\ p_1'\ \texttt{in} \\ & & \cdots \\ & & \texttt{let}\ p_n = \omega_n\ p_n'\ \texttt{in}\ v_1' \end{pmatrix}^{-1} := \begin{pmatrix} v_1' & \leftrightarrow & \texttt{let}\ p_n' = \omega_n^{-1}\ p_n\ \texttt{in} \\ & & \cdots \\ & & \texttt{let}\ p_1' = \omega_1^{-1}\ p_1\ \texttt{in}\ v_1 \end{pmatrix}.$$

▶ **Property 3** (Inversion is an involution). *For any well-typed iso ω, we have $(\omega^{-1})^{-1} = \omega$.*

Proof. By a straightforward induction on ω, notice that if $\omega = \{v_1 \leftrightarrow e_1 \mid \cdots \mid v_n \leftrightarrow e_n\}$ then by definition we swap twice the order of the `let` construction, hence recovering the original term. ◀

▶ **Lemma 4** (Inversion is well-typed). *If $\phi_1 : A_1 \leftrightarrow B_1 \ldots \phi_n : A_n \leftrightarrow B_n \vdash_\omega \omega : T$, then $\phi_1 : B_1 \leftrightarrow A_1 \ldots \phi_n : B_n \leftrightarrow A_n \vdash_\omega \omega^{-1} : T^{-1}$.* ⌟

▶ **Lemma 5** (Inversion is preserved by evaluation). *If $\omega \to \omega'$ then $\omega^{-1} \to \omega'^{-1}$.* ⌟

▶ **Theorem 6** (Semantics of isos and their inversions [8]). *For all well-typed isos $\vdash_\omega \omega : A \leftrightarrow B$, and for all well-typed values $\vdash v : A$, if $(\omega\ (\omega^{-1}\ v)) \to^* v'$ then $v = v'$.* ⌟

▶ **Example 7.** Remember that $[A] = \mu X.\mathbb{1} \oplus (A \otimes X)$. One can define the *map* operator on lists with an iso of type $(A \leftrightarrow B) \to [A] \leftrightarrow [B]$, defined as

$$\lambda\psi.\texttt{fix}\ \phi.\{[\,] \leftrightarrow [\,] \ \mid\ h :: t \leftrightarrow \texttt{let}\, h' = \psi\ h\ \texttt{in}\ \texttt{let}\, t' = \phi\ t\ \texttt{in}\ h' :: t'\}\,,$$

with the terms $[\,] = \texttt{fold}\,(\texttt{inj}_\ell\,(()))$, representing the empty list, while the head and tail of the list is represented with $h :: t = \texttt{fold}\,(\texttt{inj}_r\,(\langle h, t\rangle))$. Its inverse map^{-1} is

$$\lambda\psi.\texttt{fix}\ \phi.\{[\,] \leftrightarrow [\,] \ \mid\ h' :: t' \leftrightarrow \texttt{let}\, t = \phi\ t'\ \texttt{in}\ \texttt{let}\, h = \psi\ h'\ \texttt{in}\ h :: t\}\,.$$

Note that in the latter, the variable ψ has type $B \leftrightarrow A$. If we consider the inverse of the term $(\text{map}\ \ \omega)$ we would obtain the term $(\text{map}^{-1}\ \ \omega^{-1})$ where ω^{-1} would be of type $B \leftrightarrow A$.

▶ **Example 8** (Cantor Pairing)**.** One can encode the Cantor Pairing between $\mathbb{N} \otimes \mathbb{N} \leftrightarrow \mathbb{N}$. First recall that the type of natural number $\texttt{nat}$ is given by $\mu X.\mathbb{1} \oplus X$, then define $\overline{n}$ as the encoding of natural numbers into a closed value of type $\texttt{nat}$ as $\overline{0} = \texttt{fold}\,(\texttt{inj}_\ell\,())$ and given a variable x of type $\texttt{nat}$, its successor is $\overline{S(x)} = \texttt{fold}\,(\texttt{inj}_r\,(x))$. Omitting the $\overline{\cdot}$ operator for readability, the pairing is then defined as:

$$\omega_1 : \texttt{nat} \otimes \texttt{nat} \leftrightarrow (\texttt{nat} \otimes \texttt{nat}) \oplus \mathbb{1}$$
$$= \left\{ \begin{array}{lcl} \langle S(i), j\rangle & \leftrightarrow & \texttt{inj}_\ell\,(\langle i, S(j)\rangle) \\ \langle 0, S(S(j))\rangle & \leftrightarrow & \texttt{inj}_\ell\,(\langle S(j), 0\rangle) \\ \langle 0, S(0)\rangle & \leftrightarrow & \texttt{inj}_\ell\,(\langle 0, 0\rangle) \\ \langle 0, 0\rangle & \leftrightarrow & \texttt{inj}_r\,(()) \end{array} \right\},$$

$$\omega_2 : (\texttt{nat} \otimes \texttt{nat}) \oplus \mathbb{1} \leftrightarrow \texttt{nat}$$
$$= \left\{ \begin{array}{lcl} \texttt{inj}_\ell\,(x) & \leftrightarrow & \texttt{let}\, y = \phi\ x\ \texttt{in}\ S(y) \\ \texttt{inj}_r\,(()) & \leftrightarrow & 0 \end{array} \right\},$$

$$\text{CantorPairing} : \texttt{nat} \otimes \texttt{nat} \leftrightarrow \texttt{nat}$$
$$= \texttt{fix}\ \phi.\left\{ \begin{array}{lcl} x & \leftrightarrow & \texttt{let}\, y = \omega_1\ x\ \texttt{in} \\ & & \texttt{let}\, z = \omega_2\ y\ \texttt{in}\ z \end{array} \right\},$$

where the variable ϕ in ω_2 is the one being bound by the $\texttt{fix}$ of the CantorPairing iso. Intuitively, ω_1 realises one step of the Cantor Pairing evaluation while ω_2 checks if we reached the end of the computation and either applies a recursive call, or stops.

For instance, CantorPairing $\langle 1, 1\rangle$ will match with the first clause of ω_1, evaluating into $\texttt{inj}_\ell\ \langle 0, 2\rangle$, and then, inside ω_2 the reduction CantorPairing $\langle 0, 2\rangle$ will be triggered through the recursive call, evaluating the second clause of ω_1, reducing to $\texttt{inj}_\ell\ \langle 1, 0\rangle$, etc.

3 Expressivity

This section is devoted to assessing the expressivity of the language. To that end, we rely on Reversible Turing Machine (RTM) [1]. We describe how to encode an RTM as an iso, and prove that the iso realises the string semantics of the RTM.

3.1 Recovering duplication, erasure and manipulation of constants

Although the language is linear and reversible, since closed values are all finite, and one can build isos to encode notions of duplication, erasure, and constant manipulation thanks to partiality.

▶ **Definition 9** (Duplication)**.** We define Dup_A^S the iso of type $A \leftrightarrow A \otimes A$ which can duplicate any closed value of type A by induction on A, where S is a set of pairs of a type-variable X and an iso-variable ϕ, such that for every free-type-variable $X \subseteq A$, there exists a unique pair $(X, \phi) \in S$ for some ϕ.

The iso is defined by induction on A: $\text{Dup}_{\mathbb{1}}^S = \{() \leftrightarrow \langle (), ()\rangle\}$, and

$$\text{Dup}_{A\otimes B}^S = \left\{ \begin{array}{lcl} \langle x, y\rangle & \leftrightarrow & \texttt{let}\, \langle x_1, x_2\rangle = \text{Dup}_A^S\ \ x\ \texttt{in}\ \texttt{let}\, \langle y_1, y_2\rangle = \text{Dup}_B^S\ \ y\ \texttt{in} \\ & & \langle \langle x_1, y_1\rangle, \langle x_2, y_2\rangle\rangle \end{array} \right\};$$

- $\mathrm{Dup}^S_{A\oplus B} = \left\{ \begin{array}{lcl} \mathtt{inj}_\ell\ (x) & \leftrightarrow & \mathtt{let}\ \langle x_1, x_2\rangle = \mathrm{Dup}^S_A\ \ x\ \mathtt{in}\ \langle \mathtt{inj}_\ell\ (x_1), \mathtt{inj}_\ell\ (x_2)\rangle \\ \mathtt{inj}_r\ (y) & \leftrightarrow & \mathtt{let}\ \langle y_1, y_2\rangle = \mathrm{Dup}^S_B\ \ y\ \mathtt{in}\ \langle \mathtt{inj}_r\ (y_1), \mathtt{inj}_r\ (y_2)\rangle \end{array} \right\}$;
- If $(X, \_) \notin S$: $\mathrm{Dup}^S_{\mu X.A} = \mathtt{fix}\ \phi. \left\{ \begin{array}{c} \mathtt{fold}\ (x) \leftrightarrow \mathtt{let}\ \langle x_1, x_2\rangle = Dup^{S\cup\{(X,\phi)\}}_{A[\mu X.A/X]}\ x\ \mathtt{in} \\ \langle \mathtt{fold}\ (x_1), \mathtt{fold}\ (x_2)\rangle \end{array} \right\}$;
- If $(X, \phi) \in S$: $\mathrm{Dup}^S_{\mu X.A} = \{x \leftrightarrow \mathtt{let}\ \langle x_1, x_2\rangle = \phi\ x\ \mathtt{in}\ \langle x_1, x_2\rangle\}$.

Remember that bound variables are assumed distinct following Barendregt's convention, allow for the well-definition of the isos above.

▶ **Lemma 10** (Properties of Duplication). *Given a closed type A, then $\mathrm{Dup}^\emptyset_A$ is well-defined, and the iso $\mathrm{Dup}^\emptyset_A$ is well typed of type $A \leftrightarrow A \otimes A$.* ⌟

▶ **Lemma 11** (Semantics of Duplication). *Given a closed type A and a closed value v of type A, then $\mathrm{Dup}^\emptyset_A\ \ v \to^* \langle v_1, v_2\rangle$ and $v = v_1 = v_2$.* ⌟

▶ **Definition 12** (Constant manipulation). We define $\mathrm{erase}_v : A \otimes \Sigma^T \leftrightarrow A$ which erases its second argument when its value is v as $\{\langle x, v\rangle \leftrightarrow x\}$. Reversed, it turns any x into $\langle x, v\rangle$.

3.2 Definition of Reversible Turing Machine

▶ **Definition 13** (Reversible Turing Machine [1]). Let $M = (Q, \Sigma, \delta, b, q_s, q_f)$ be a Turing Machine, where Q is a set of states, $\Sigma = \{b, a_1, \ldots, a_n\}$ is a finite set of tape symbols (in the following, a_i and b always refer to elements of Σ), $\delta \subseteq \Delta = (Q \times [(\Sigma \times \Sigma) \cup \{\leftarrow, \downarrow, \rightarrow\}] \times Q)$ is a partial relation defining the transition relation such that there must be no transitions leading out of q_f nor into q_s, b a blank symbol and q_s and q_f the initial and final states. We say that M is a *Reversible Turing Machine* (RTM) if it is:

- *forward* deterministic: for any two distinct pairs of triples (q_1, a_1, q_1') and (q_2, a_2, q_2') in δ, if $q_1 = q_2$ then $a_1 = (s_1, s_1')$ and $a_2 = (s_2, s_2')$ and $s_1 \neq s_2$.
- *Backward* deterministic: for any two distinct pairs of triples (q_1, a_1, q_1') and (q_2, a_2, q_2') in δ, if $q_1' = q_2'$ then $a_1 = (s_1, s_1')$ and $a_2 = (s_2, s_2')$ and $s_1' \neq s_2'$.

▶ **Definition 14** (Configurations [1]). A *configuration* of a RTM is a tuple $(q, (l, s, r)) \in \mathrm{Conf} = Q \times (\Sigma^* \times \Sigma \times \Sigma^*)$ where q is the internal state, l, r are the left and right parts of the tape (as string) and $s \in \Sigma$ is the current symbol being scanned. A configuration is *standard* when the cursor is on the immediate left of a finite, blank-free string $s \in (\Sigma \setminus \{b\})^*$ and the rest is blank, i.e. it is in configuration $(q, (\epsilon, b, s))$ for some q, where ϵ is the empty string, representing an infinite sequence of blank symbols b.

▶ **Definition 15** (RTM Transition [1]). An RTM M in configuration $C = (q, (l, s, r))$ goes to a configuration $C' = (q', (l', s', r'))$, written $T \vdash C \rightsquigarrow C'$ in a single step if there exists a transition $(q, a, q') \in \delta$ where a is either (s, s'), and then $l = l'$ and $r = r'$ or $a \in \{\leftarrow, \downarrow, \rightarrow\}$, and we have for the case $a = \leftarrow$: $l' = l \cdot s$ and for $r = x \cdot r_2$ we have $s' = x$ and $r' = r_2$, similarly for the case $a = \rightarrow$ and for the case $a = \downarrow$ we have $l' = l$ and $r' = r$ and $s = s'$.

The semantics of an RTM is given on *standard configurations* of the form $(q, (\epsilon, b, s))$ where q is a state, ϵ is the finite string standing for a blank-filled tape, and s is the blank-free, finite input of the RTM.

▶ **Definition 16** (String Semantics [1]). The semantics of a RTM M, written $\mathrm{Sem}(M)$ is defined on standards configurations and is given by the set $\mathrm{Sem}(M) = \{(s, s') \in ((\Sigma\setminus\{b\})^* \times (\Sigma\setminus\{b\})^*) \mid M \vdash (q_s, (\epsilon, b, s)) \rightsquigarrow^* (q_f, (\epsilon, b, s'))\}$.

▶ **Theorem 17** (Properties of RTM [1]). *For all RTM M,* $\mathrm{Sem}(M)$ *is the graph of an injective function. Conversely, all injective computable functions (on a tape) are realisable by a RTM. Finally, any Turing Machine can be simulated by a Reversible Turing Machine.* ⌟

3.3 Encoding RTMs as Isos

A RTM configuration is a set-based construction that we can model using the type constructors available in our language. Because the transition relation δ is backward and forward deterministic, it can be encoded as an iso. Several issues need to be dealt with; we discuss them in this section.

Encoding configurations. The set of states $Q = \{q_1, \ldots, q_n\}$ is modeled with the type $Q^T = \mathbb{1} \oplus \cdots \oplus \mathbb{1}$ (n times). The encoding of the state q_i is then a closed value q_i^T. They are pairwise orthogonal. The set Σ of tape symbols is represented similarly by $\Sigma^T = \mathbb{1} \oplus \cdots \oplus \mathbb{1}$, and the encoding of the tape symbol a is a^T. We then define the type of configurations in the obvious manner: a configuration $C = (q, (l, s, r))$ corresponds to a closed value $\mathrm{isos}(C)$ of type $Q^T \otimes ([\Sigma^T] \otimes \Sigma^T \otimes [\Sigma^T])$.

▶ **Definition 18** (Encoding of Configurations). We define the type of configurations as $C^T = (Q^T \otimes ([\Sigma^T] \otimes \Sigma \otimes [\Sigma^T]))$. Given a configuration $C = (q, ((\epsilon, a_1, \ldots, a_n), a, (a'_1, \ldots, a'_m, \epsilon)))$, it is encoded as $\mathrm{isos}(C) = (q^T, ([a_n^T, \ldots, a_1^T], a^T, [a_1'^T, \ldots, a_m'^T]))$. For example, the standard configuration $C = (q_s, (\epsilon, b, [a_1, \ldots, a_n]))$ is represented as $\mathrm{isos}(C) = (q_s^T, ([], b^T, [a_1^T, \ldots, a_n^T]))$.

Encoding the transition relation δ. A limitation of our language is that every subcomputation has to be reversible and does not support infinite data structures such as streams. In the context of RTMs, the empty string ϵ is identified with an infinite string of blank symbols. If this can be formalised in set theory, in our limited model, we cannot emit blank symbols out of thin air without caution.

In order to simulate an infinite amount of blank symbols on both sides of the tape during the evaluation, we provide an iso that grows the size of the two tapes on both ends by blank symbols at each transition step. The iso `growth` is shown in Table 5. It is built using three auxiliary functions, written in a Haskell-like notation. `len` sends a closed value $[v_1, \ldots, v_n]$ to $\langle [v_1, \ldots, v_n], \overline{n} \rangle$. `snoc′` sends $\langle [v_1, \ldots, v_n], v, \overline{n} \rangle$ to $\langle [v_1, \ldots, v_n, v], v, \overline{n} \rangle$. `snoc` sends $\langle [v_1, \ldots, v_n], v \rangle$ to $\langle [v_1, \ldots, v_n, v], v \rangle$. Finally, `growth` sends $\langle [a_1^T, \ldots, a_n^T], [a_1'^T, \ldots, a_m'^T] \rangle$ to $\langle [a_1^T, \ldots, a_n^T, b^T], [a_1'^T, \ldots, a_m'^T, b^T] \rangle$.

Now, given a RTM $M = (Q, \Sigma, \delta, b, q_s, q_f)$, a relation $(q, r, q') \in \delta$ is encoded as a clause between values $\mathrm{iso}(q, r, q') = v_1 \leftrightarrow v_2$ of type $C^T \leftrightarrow C^T$. These clauses are defined by case analysis on r as follows. When x, x', z, y and y' are variables:

- $\mathrm{iso}(q, \rightarrow, q') = (q^T, (x', z, y :: y')) \leftrightarrow \texttt{let}\ (l, r) = \texttt{growth}\ (x', y')\ \texttt{in}\ (q'^T, (z :: l, y, r))$,
- $\mathrm{iso}(q, \leftarrow, q') = (q^T, (x :: x', z, y')) \leftrightarrow \texttt{let}\ (l, r) = \texttt{growth}\ (x', y')\ \texttt{in}\ (q'^T, (l, x, z :: r))$,
- $\mathrm{iso}(q, \downarrow, q') = (q^T, (x', z, x')) \leftrightarrow \texttt{let}\ (l, r) = \texttt{growth}\ (x', y')\ \texttt{in}\ (q'^T, (l, z, r))$,
- $\mathrm{iso}(q, (s, s'), q') = (q^T, (x', s^T, y')) \leftrightarrow \texttt{let}\ (l, r) = \texttt{growth}\ (x', y')\ \texttt{in}\ (q'^T, (l, s'^T, r))$.

The encoding of the RTM M is then the iso $\mathrm{isos}(M)$ whose clauses are the encoding of each rule of the transition relation δ, of type $\mathrm{Conf}^T \leftrightarrow \mathrm{Conf}^T$.

Encoding successive applications of δ. The transition δ needs to be iterated until the final state is reached. This behavior can be emulated in our language using the iso `It`, defined in Table 5. The iso $\texttt{It}\,\omega$ is typed with $(A \leftrightarrow A \otimes \texttt{nat})$. Fed with a value of type A, it iterates ω until `ff` is met. It then returns the result together with the number of iterations.

Table 5 Some useful isos for the encoding.

$$
\begin{array}{l}
\texttt{len} : [A] \leftrightarrow [A] \otimes \texttt{nat} \\
\texttt{len}\,[\,] \quad \leftrightarrow ([\,], 0) \\
\texttt{len}\,h :: t \leftrightarrow \texttt{let}\,(t', n) = \texttt{len}\ t\ \texttt{in} \\
\qquad\qquad (h :: t', S(n))
\end{array}
$$

$$
\begin{array}{l}
\texttt{snoc}' : [A] \otimes A \otimes \texttt{nat} \leftrightarrow [A] \otimes A \otimes \texttt{nat} \\
\texttt{snoc}'\,([\,], x, 0) \qquad \leftrightarrow \texttt{let}\,(x_1, x_2) = \mathrm{Dup}_A^{\emptyset}\ x\ \texttt{in} \\
\qquad\qquad\qquad\qquad ([x_1], x_2, 0) \\
\texttt{snoc}'\,(h :: t, x, S(n)) \leftrightarrow \texttt{let}\,(t', x', n') = \texttt{snoc}'(t, x, n)\ \texttt{in} \\
\qquad\qquad\qquad\qquad (h :: t', x', S(n'))
\end{array}
$$

$$
\begin{array}{l}
\texttt{snoc} : [A] \otimes A \leftrightarrow [A] \otimes A \\
\texttt{snoc}\,(x, y) \leftrightarrow \texttt{let}\,(x', n) = \texttt{len}\ x\ \texttt{in} \\
\qquad\qquad\qquad \texttt{let}\,(x'', y', n') = \texttt{snoc}'\ (x', y, n)\ \texttt{in} \\
\qquad\qquad\qquad \texttt{let}\,n'' = \{x \ \leftrightarrow \ Sx\}\ n'\ \texttt{in} \\
\qquad\qquad\qquad \texttt{let}\,z = \texttt{len}^{-1}\ (x'', n'')\ \texttt{in}\ (z, y')
\end{array}
$$

$$
\begin{array}{l}
\texttt{growth} : [\Sigma^T] \otimes [\Sigma^T] \leftrightarrow [\Sigma^T] \otimes [\Sigma^T] \\
\texttt{growth}\,(l, r) \leftrightarrow \texttt{let}\,\langle l', b_1\rangle = \texttt{snoc}\langle l, b^T\rangle\ \texttt{in} \\
\qquad\qquad\qquad\quad \texttt{let}\,\langle r', b_2\rangle = \texttt{snoc}\langle r, b^T\rangle\ \texttt{in} \\
\qquad\qquad\qquad\quad \texttt{let}\,l'' = \texttt{erase}_b\langle l', b_1\rangle\ \texttt{in} \\
\qquad\qquad\qquad\quad \texttt{let}\,r'' = \texttt{erase}_b\langle r', b_2\rangle\ \texttt{in}\ (l'', r'')
\end{array}
$$

$$
\begin{array}{l}
\texttt{It} : (A \leftrightarrow A \otimes (\mathbb{1} \oplus \mathbb{1})) \to (A \leftrightarrow A \otimes \texttt{nat}) \\
\texttt{It}\psi\ x \leftrightarrow \texttt{let}\ y = \psi\ x\ \texttt{in} \\
\qquad\qquad \texttt{let}\ z = \left\{ \begin{array}{l} (y, \texttt{tt}) \leftrightarrow \texttt{let}\,(z, n) = (\texttt{It}\,\psi)\ y\ \texttt{in}\ (z, S\ n) \\ (y, \texttt{ff}) \leftrightarrow (y, 0) \end{array} \right\}\ y\ \texttt{in}\ z
\end{array}
$$

$$
\begin{array}{l}
\texttt{rmBlank} : [\Sigma] \leftrightarrow [\Sigma] \otimes \mathbb{N} \\
\texttt{rmBlank}\,[] \qquad \leftrightarrow ([], 0) \\
\texttt{rmBlank}\,b^T :: t \ \leftrightarrow \texttt{let}\,(t', n) = \texttt{rmBlank}\ t\ \texttt{in}\ (t', S(n)) \\
\texttt{rmBlank}\,a_1^T :: t \leftrightarrow ((a_1^T :: t), 0) \\
\vdots \qquad\quad \vdots \qquad\quad \vdots\ \vdots \\
\texttt{rmBlank}\,a_n^T :: t \leftrightarrow ((a_n^T :: t), 0)
\end{array}
$$

$$
\begin{array}{l}
\texttt{rev}_{\text{aux}} : [A] \otimes [A] \leftrightarrow [A] \otimes [A] \\
\texttt{rev}_{\text{aux}}\,([], y) \qquad \leftrightarrow ([], y) \\
\texttt{rev}_{\text{aux}}\,(h :: t, y) \leftrightarrow \texttt{let}\,(h_1, h_2) = \mathrm{Dup}_A^{\emptyset}\ \ h\ \texttt{in} \\
\qquad\qquad\qquad\qquad\quad \texttt{let}\,(t_1, t_2) = \phi(t, h_2 :: y)\ \texttt{in} \\
\qquad\qquad\qquad\qquad\quad (h_1 :: t_1, t_2)
\end{array}
$$

$$
\begin{array}{l}
\texttt{rev} : [A] \leftrightarrow [A] \otimes [A] \\
\texttt{rev} = \{x \leftrightarrow \texttt{let}\,(t_1, t_2) = \texttt{rev}_{\text{aux}}\ (x, [])\ \texttt{in}\ (t_1, t_2)\}
\end{array}
$$

$$
\begin{array}{l}
\texttt{cleanUp} : C^T \otimes \texttt{nat} \leftrightarrow C^T \otimes \texttt{nat} \otimes \texttt{nat} \otimes \texttt{nat} \otimes [\Sigma^T] \\
\texttt{cleanUp}\,((x, (l, y, r)), n) \leftrightarrow \texttt{let}\,(l', n_1) = \texttt{rmBlank}\ l\ \texttt{in} \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \texttt{let}\,(r_{\text{ori}}, r_{\text{rev}}) = \texttt{rev}\ r\ \texttt{in} \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \texttt{let}\,(r', n_2) = \texttt{rmBlank}\ r_{\text{rev}}\ \texttt{in} \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad ((x, (l', y, r')), n, n_1, n_2, r_{\text{ori}})
\end{array}
$$

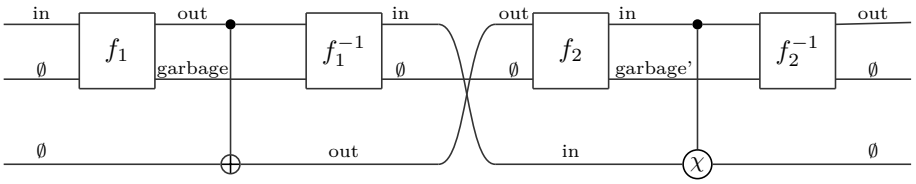


Figure 1 Reversibly removing additional garbage from some process.

To iterate iso(M), we then only need to modify iso to return a boolean stating whether q_f was met. This can be done straightforwardly, yielding an iso $\text{isos}_{\mathbb{B}}(M))$ of type $\text{Conf}^T \leftrightarrow \text{Conf}^T \otimes (\mathbb{1} \oplus \mathbb{1})$. With such an iso, given M be a RTM such that $M \vdash (q_s, (\epsilon, b, s)) \rightsquigarrow^{n+1} (q_f, (\epsilon, b, (a_1, \ldots, a_n)))$, then $\texttt{It}(\text{isos}_{\mathbb{B}}(M))\ (q_s^T, ([b^T], b^T, s^T))$ reduces to the encoding term $((q_f^T, ([b^T, \ldots, b^T], b^T, [a_1^T, \ldots, a_n^T, b^T, \ldots, b^T])), \overline{n})$. If it were not for the additional blank tape elements, we would have the encoding of the final configuration.

Recovering a canonical presentation. Removing blank states at the *beginning* of a list is easy: for instance, it can be done with the iso rmBlank, shown in Table 5. Cleaning up the tail of the list can then be done by reverting the list, using, e.g. rev in the same table. By abuse of notation, we use constants in some patterns: an exact representation would use Definition 12. Finally, we can define the operator `cleanUp`, solving the issue raised in the previous paragraph. In particular, given a RTM M and an initial configuration C such that $M \vdash C \rightsquigarrow C' = (q, (\epsilon, b, (a_1, \ldots, a_n)))$, then we have that $\texttt{cleanUp}\ (\texttt{It}(\text{isos}_{\mathbb{B}}(M))C^T) \rightarrow^* ((q^T, ([], b^T, [a_1^T, \ldots, a_n^T])), v)$, where v is of type $\texttt{nat} \otimes \texttt{nat} \otimes \texttt{nat} \otimes [\Sigma^T]$. If we want to claim that we indeed capture the operational behaviour of RTMS, we need to get rid of this value v.

Getting rid of the garbage. To discard this value v, we rely on Bennett's trick [4], shown in Figure 1. Given two Turing machines f_1 and f_2 and some input `in` such that if $f_1(\texttt{in}) = \texttt{out} \otimes \texttt{garbage}$ and $f_2(\texttt{out}) = \texttt{in} \otimes \texttt{garbage}'$, then the process consists of taking additional tapes in the Turing Machine in order to reversibly duplicate (represented by the $\oplus$) or reversibly erase some data (represented by the χ) in order to recover only the output of f_1, without any garbage.

Given an iso $\omega \colon A \leftrightarrow B \otimes C$ and $\omega' \colon B \leftrightarrow A \otimes C'$ where C, C' represent garbage, we can build an iso from $A \leftrightarrow B$ as follows, where the variables x, y, z (and their indices) respectively correspond to the first, second, and third wire of Figure 1. This operator makes use of the iso Dup discussed in Section 3.1.

$$\begin{aligned}\texttt{GarbRem}(\omega, \omega')\ x_1 \quad \leftrightarrow \quad & \texttt{let}\ \langle x_2, y\rangle = \omega\ x_1\ \texttt{in}\ \texttt{let}\ \langle x_3, z\rangle = \text{Dup}_B^{\emptyset}\ x_2\ \texttt{in} \\ & \texttt{let}\ x_4 = \omega^{-1}\ \langle x_3, y\rangle\ \texttt{in}\ \texttt{let}\ \langle z_2, y_2\rangle = \omega'\ z\ \texttt{in} \\ & \texttt{let}\ z_3 = (\text{Dup}_B^{\emptyset})^{-1}\ \langle z_2, x_4\rangle\ \texttt{in}\ \texttt{let}\ z_4 = \omega'^{-1}\ \langle z_3, y_2\rangle\ \texttt{in}\ z_4.\end{aligned}$$

▶ **Theorem 19** (Capturing the exact semantics of a RTM). *For all RTM M with standard configurations $C = (q_s, (\epsilon, b, s))$ and $C' = (q_f, (\epsilon, b, s'))$ such that $M \vdash C \rightsquigarrow^* C'$, we have*

$$\texttt{GarbRem}(\texttt{cleanUp}(\texttt{It}(\text{isos}_{\mathbb{B}}(M))), \texttt{cleanUp}(\texttt{It}(\text{isos}_{\mathbb{B}}(M^{-1})))\ \ \text{isos}(C) \rightarrow^* \text{isos}(C')$$

The behavior of RTMs is thus captured by the language. ⌟

4 Categorical Background

We aim at providing a denotational semantics for the programming language introduced above, meaning a mathematical interpretation abstract to the syntax. Our approach is categorical, in the spirit of many others before us. Programs are compositional by design, making it natural to interpret in a framework ruled by compositionality. Types are usually interpreted as objects in a category $\mathcal{C}$, and terms as morphisms in this category. We have seen that the main feature of our programming language is reversibility and its terms can be seen as partial isomorphisms, or partial injections. We want this property to be carried on the interpretation, and we present in this section the proper categories to do so. The category of sets and partial injective functions, written **PInj**, will be the recurring example throughout this section to help the intuition.

4.1 Join inverse rig category

The axiomatisation of join inverse rig categories gives the conditions for the morphisms of a category to be *partial injections*. First, the notion of restriction allows to capture the *actual* domain of a morphism through a partial identity function. Historically, *inverse* categories [27] were introduced before *restriction* categories, but the latter are more convenient to introduce the subject.

▶ **Definition 20** (Restriction [9])**.** A restriction structure is an operator that maps each morphism $f : A \to B$ to a morphism $\overline{f} : A \to A$ such that for all g and h such that the domain of g is A and the domain of h is B we have $f \circ \overline{f} = f$, $\overline{f} \circ \overline{g} = \overline{g} \circ \overline{f}$, $\overline{f \circ \overline{g}} = \overline{f} \circ \overline{g}$ and $\overline{h} \circ f = f \circ \overline{h \circ f}$. A morphism f is said to be *total* if $\overline{f} = 1_A$. A category with a restriction structure is called a *restriction category*. A functor $F : \mathcal{C} \to \mathcal{D}$ is a *restriction functor* if $\overline{F(f)} = F\left(\overline{f}\right)$ for all morphism f of $\mathcal{C}$. The definition is canonically extended to bifunctors. When unambiguous, we write gf for the composition $g \circ f$.

▶ **Example 21.** Given sets A, B and a partial function $f : A \to B$ defined on $A' \subseteq A$ and undefined on $A \setminus A'$, the restriction of f is $\overline{f} : A \to A$, the identity on $A' \subseteq A$ and undefined on $A \setminus A'$. This example shows that **PInj** is a restriction category.

To interpret reversibility, we need to introduce a notion of reversed process, a process that exactly reverses another process. This is given by a generalised notion of inverse.

▶ **Definition 22** (Inverse category [24])**.** An *inverse category* is a restriction category where all morphisms are partial isomorphisms; meaning that for $f : A \to B$, there exists a unique $f^\circ : B \to A$ such that $f^\circ \circ f = \overline{f}$ and $f \circ f^\circ = \overline{f^\circ}$.

▶ **Example 23.** In **PInj**, let us consider the partial function $f : \{0, 1\} \to \{0, 1\}$ as $f(0) = 1$ and undefined on 1. Its restriction $\overline{f}$ is undefined on 1 also but $\overline{f}(0) = 0$. Its *inverse* f° is undefined on 0 and such that $f^\circ(1) = 0$.

The example above generalises and **PInj** is an actual inverse category. Even more, it is *the* inverse category: [27] proves that every locally small inverse category is isomorphic to a subcategory of **PInj**.

▶ **Definition 24** (Restriction compatible [24])**.** Two morphisms $f, g : A \to B$ in a restriction category $\mathcal{C}$ are restriction compatible if $f\overline{g} = g\overline{f}$. The relation is written $f \smile g$. If $\mathcal{C}$ is an inverse category, they are inverse compatible if $f \smile g$ and $f^\circ \smile g^\circ$, noted $f \asymp g$. A set S of morphisms of the same type $A \to B$ is restriction compatible (*resp.* inverse compatible) if all elements of S are pairwise restriction compatible (*resp.* inverse compatible).

▶ **Definition 25** (Partial order [9]). Let $f, g : A \to B$ be two morphisms in a restriction category. We then define $f \leq g$ as $g\overline{f} = f$.

▶ **Definition 26** (Joins [16]). A restriction category $\mathcal{C}$ is equipped with joins if for all restriction compatible sets S of morphisms $A \to B$, there exists $\bigvee_{s\in S} s : A \to B$ morphism of $\mathcal{C}$ such that, whenever $t\colon A \to B$ and whenever for all $s \in S$, $s \leq t$, $s \leq \bigvee_{s\in S} s$, $\bigvee_{s\in S} s \leq t$, $\overline{\bigvee_{s\in S} s} = \bigvee_{s\in S} \overline{s}$, $f \circ \left(\bigvee_{s\in S} s\right) = \bigvee_{s\in S} fs$, $\left(\bigvee_{s\in S} s\right) \circ g = \bigvee_{s\in S} sg$. Such a category is called a *join restriction category*. An inverse category with joins is called a *join inverse category*.

Building up from Definition 20, a *join restriction functor* is a restriction functor that preserves all thus constructed joins.

▶ **Definition 27** (Zero [24]). Since $\emptyset \subseteq \mathrm{Hom}_{\mathcal{C}}(A, B)$, and since all of its elements are restriction compatible, there exists a morphism $0_{A,B} \doteq \bigvee_{s\in\emptyset} s$, called *zero*. It satisfies the following equations: $f0 = 0$, $0g = 0$, $0^{\circ}_{A,B} = 0_{B,A}$, $\overline{0_{A,B}} = 0_{A,A}$.

▶ **Definition 28** (Restriction Zero). *A restriction category $\mathcal{C}$ has a* restriction zero *object 0 iff for all objects A and B, there exists a unique morphism $0_{A,B}\colon A \to B$ that factors through* 0 *and satisfies $\overline{0_{A,B}} = 0_{A,A}$.*

▶ **Definition 29** (Disjointness tensor [13]). An inverse category $\mathcal{C}$ is said to have a *disjointness tensor* if it is equipped with a symmetric monoidal restriction bifunctor $. \oplus . : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$, with as unit a restriction zero 0 and morphisms $\iota_l : A \to A \oplus B$ and $\iota_r : B \to A \oplus B$ that are total, jointly epic, and such that their inverses are jointly monic and $\overline{\iota_l^{\circ}}\ \overline{\iota_r^{\circ}} = 0_{A\oplus B}$.

▶ **Definition 30** ([25]). Let us consider a join inverse category equipped with a symmetric monoidal tensor product $(\otimes, 1)$ and a disjointness tensor $(\oplus, 0)$ that are join preserving, and such that there are isomorphisms $\delta_{A,B,C}\colon A \otimes (B \oplus C) \to (A \otimes B) \oplus (A \otimes C)$ and $\nu_A\colon A \otimes 0 \to 0$. This is called a *join inverse rig category*.

4.2 DCPO-category

We use the vocabulary of enriched category theory to shorten the discussion in this section. The notions of enrichment required to understand the semantics later is basic and should not frighten the reader. Categories in computer science are usually *locally small*, meaning that given two objects A and B, there is a *set* of morphisms $A \to B$. Enrichment is the study of the structure of those sets of morphisms, which could be vector spaces or topological spaces for example, more details can be found in [28, 29, 33]. It turns out that homsets in join inverse rig categories are dcpos – directed-complete partial orders, i.e. a partial-ordered set with all directed joins. This allows us to consider fixpoints in homsets. **DCPO** is the category of directed complete partial orders and Scott-continuous functions – monotone functions preserving joins. Dcpos are often used for the denotational interpretation of different sorts of λ-calculi, and more generally, to interpret recursive functions or indefinite loops.

▶ **Definition 31** ([12]). A category enriched over **DCPO**, also called a **DCPO**-category, is a locally small category whose hom-sets are directly partial ordered and where composition is a continuous operation (*i.e.* a morphism in **DCPO**).

It is proven in [24] that a join inverse category can be considered enriched in **DCPO** without loss of generality.

▶ **Lemma 32.** *Let $\mathcal{C}$ be a join inverse rig category. The functors: $- \otimes -\colon \mathcal{C} \times \mathcal{C} \to \mathcal{C}$, $- \oplus -\colon \mathcal{C} \times \mathcal{C} \to \mathcal{C}$, $-^{\circ}\colon \mathcal{C}^{op} \to \mathcal{C}$ are* **DCPO***-functors, meaning that they preserve the dcpo structure of homsets.*

4.3 Compactness

Inductive data types are written in the syntax as some least fixed point. As said earlier, types are represented as objects in the category, and thus a type judgement is an object mapping, or rather an endofunctor. Here, we show how to consider fixed points of endofunctors in our categorical setting.

▶ **Definition 33** (Initial Algebra). Given an endofunctor $F\colon \mathcal{C} \to \mathcal{C}$, an F-algebra is a pair of an object A and a morphism $f\colon FA \to A$. F-algebras form a category with F-algebras homomorphisms. An initial F-algebra is an initial object in the category of F-algebras.

▶ **Theorem 34** (Lambek's theorem). *Given an endofunctor $F : \mathcal{C} \to \mathcal{C}$ and an F-initial algebra $(X, \alpha\colon FX \to X)$, α is an isomorphism.* ⌟

With Lambek's theorem, we know that an initial algebra provides an object X such that $X \cong FX$; X is a fixed point of the endofunctor F, as requested. The existence of such fixed points is given by the next theorem [12, Corollary 7.2.4].

▶ **Definition 35** (Ep-pair). *Given a* **DCPO***-category $\mathcal{C}$, a morphism $e\colon X \to Y$ in $\mathcal{C}$ is called an* embedding *if there exists a morphism $p\colon Y \to X$ such that $p \circ e = \mathrm{id}_X$ and $e \circ p \leq \mathrm{id}_Y$. The morphisms e and p form an* embedding-projection pair (e, p)*, also called* ep-pair*.*

We recall that an *ep-zero* [12, Definition 7.1.1], is an initial object such that every morphism with it as source is an embedding, and is also a terminal object such that every morphism with it as target is a projection.

▶ **Theorem 36.** *A* **DCPO***-category with an ep-zero and colimits of ω-chains of embeddings is parametrised* **DCPO***-algebraically ω-compact; meaning that for every* **DCPO***-functor $F\colon \mathcal{C} \times \mathcal{D} \to \mathcal{D}$, there is a pair consisting of a* **DCPO***-functor $F^{\natural}\colon \mathcal{C} \to \mathcal{D}$ and an indexed family $\alpha^F = \{\alpha^F_A\colon F(A, F^{\natural}A) \to F^{\natural}A\}$ of initial $F(A, -)$-algebras. This pair is called a* parametrised *initial algebra.* ⌟

The hypotheses of the theorem above are verified by the categories we want to work with, without loss of generality.

▶ **Proposition 37** ([24]). *Any join inverse rig category can be faithfully embedded in a rig join inverse category with colimits of ω-chains of embeddings.* ⌟

5 Denotational semantics

We now show how to build a denotational semantics for the language we presented thus far. The semantics is akin to the one presented in [7] but with extra structure to handle inductive types and recursive functions. While the semantics is sound and adequate w.r.t. a notion of operational equivalence between terms, the main interest of the semantics rest in showing that, given some RTM M whose semantics is a function f, we show that the semantics of $\mathrm{isos}(M)$ is the same as f. This would provide us with a formal proof that any computable reversible function can be captured by an iso.

Types. Let us consider $\mathcal{C}$ a join inverse rig category (Definition 30). We can assume without loss of generality that $\mathcal{C}$ satisfies the hypothesis of Theorem 36. In order to deal with open types, we make use an auxiliary judgement for types, of the form $X_1, \ldots, X_n \vDash A$, where $\{X_i\}_i$ is a subset of the free type variables appearing in A. We interpret this

kind of judgement as a **DCPO**-functor $\mathcal{C}^{|\Theta|} \to \mathcal{C}$ written $[\![\Theta \vDash A]\!]$. This can be formally defined as a (simple) inductive relation, and the semantics is defined similarly to what is done in [12, 21]. $[\![\Theta \vDash \mathbb{1}]\!]$ is the constant functor that maps to the tensor product unit. $[\![\Theta, X \vDash X]\!]$ is a projection. The other judgements are obtained by induction: if $[\![\Theta \vDash A]\!] = f$ and $[\![\Theta \vDash B]\!] = g$, then $[\![\Theta \vDash A \oplus B]\!] = \oplus \circ \langle f, g \rangle$ and $[\![\Theta \vDash A \otimes B]\!] = \otimes \circ \langle f, g \rangle$. Finally, $[\![\Theta \vDash \mu X.A]\!] = ([\![\Theta, X \vDash A]\!])^{\curlyvee}$. All this is summed up in Table 6.

Table 6 Interpretation of types.

$$
\begin{aligned}
[\![\Theta \vDash A]\!] &: \mathcal{C}^{|\Theta|} \to \mathcal{C} \\
[\![\Theta, X \vDash X]\!] &= \Pi \\
[\![\Theta \vDash I]\!] &= K_1 \\
[\![\Theta \vDash A \oplus B]\!] &= \oplus \circ \langle [\![\Theta \vDash A]\!], [\![\Theta \vDash B]\!] \rangle \\
[\![\Theta \vDash A \otimes B]\!] &= \otimes \circ \langle [\![\Theta \vDash A]\!], [\![\Theta \vDash B]\!] \rangle \\
[\![\Theta \vDash \mu X.A]\!] &= ([\![\Theta, X \vDash A]\!])^{\curlyvee}
\end{aligned}
$$

Lemma 32 and Theorem 36 ensure that this is well-defined. For closed types, we have $[\![\mathbb{1}]\!] = 1$, $[\![A \oplus B]\!] = [\![A]\!] \oplus [\![B]\!]$, $[\![A \otimes B]\!] = [\![A]\!] \otimes [\![B]\!]$ and $[\![\mu X.A]\!] \cong [\![A[\mu X.A/X]]\!]$. Ground iso types are represented by dcpos of morphisms in $\mathcal{C}$, written $[\![A \leftrightarrow B]\!] = \mathrm{Hom}_{\mathcal{C}}([\![A]\!], [\![B]\!])$. The type of iso functions $T_1 \to T_2$ is interpreted by the dcpo of Scott continuous maps between the two dcpos $[\![T_1]\!]$ and $[\![T_2]\!]$, written $[[\![T_1]\!] \to [\![T_2]\!]]$. The terms used to build isos are dependent in two contexts: variables in Δ and isos in Ψ. In general, if $\Delta = x_1 : A_1, \ldots, x_m : A_m$ and $\Psi = \phi_1 : T_1, \ldots, \phi_n : T_n$, then we set $[\![\Delta]\!] = [\![A_1]\!] \otimes \cdots \otimes [\![A_m]\!]$ and $[\![\Psi]\!] = [\![T_1]\!] \times \cdots \times [\![T_2]\!]$, with $\otimes$ being the monoidal product in $\mathcal{C}$ and $\times$ the cartesian product in **DCPO**.

Terms. A well-formed term judgement $\Psi; \Delta \vdash t : A$ has for semantics a Scott continuous map $[\![\Psi; \Delta \vdash t : A]\!] \in \mathbf{DCPO}([\![\Psi]\!], \mathcal{C}([\![\Delta]\!], [\![A]\!]))$, defined as in Table 7 when $g \in [\![\Psi]\!]$. All this is well-defined in **DCPO** provided that $[\![\Psi \vdash_\omega \omega : A \leftrightarrow B]\!]$ is. This last point is the focus of the next section.

▶ **Lemma 38.** *Given two judgements $\Psi; \Delta_1 \vdash t_1 : A$ and $\Psi; \Delta_2 \vdash t_2 : A$, such that $t_1 \perp t_2$, we have for all $g \in [\![\Psi]\!]$ the equality $[\![t_1]\!](g)^\circ \circ [\![t_2]\!](g) = 0_{[\![\Delta_2]\!],[\![\Delta_1]\!]}$.* ⌟

Isos. Isos do only depend on function variables, but they are innately morphisms, so their denotation will be similar to terms – a Scott continuous map. We define the denotation of an iso by induction on the typing rules. The interpretation of an iso-variable is direct, it is the projection on the last component. The interpretations of evaluations and λ-abstractions are usual in a cartesian closed category, in our case, **DCPO**. All the rules apart for the iso-abstraction are found in Table 7. The remaining rule, building an iso abstraction $\{v_i \leftrightarrow e_i\}_{i \in I}$, needs more details.

▶ **Lemma 39.** *Given a well-formed iso abstraction $\Psi \vdash_\omega \{v_i \leftrightarrow e_i\}_{i \in I} : A \leftrightarrow B$, for all $g \in [\![\Psi]\!]$, the morphisms in $\mathcal{C}$ given by $[\![\Psi; \Delta_i \vdash e_i : B]\!](g) \circ [\![\Psi; \Delta_i \vdash v_i : A]\!](g)^\circ$, with $i \in I$ are pairwise inverse compatible.* ⌟

Each clause $v_i \leftrightarrow e_i$ of an iso abstraction is given an interpretation $[\![e_i]\!] \circ [\![v_i]\!]^\circ$. The previous lemma shows that in the case of an iso abstraction, the interpretations of all clauses can be joined (in the sense of Definition 26). This join also generalises to the join in **DCPO** as shown by the lemma below.

Table 7 Denotational semantics of the language in a join inverse rig **DCPO**-category.

$$
\begin{aligned}
[\![\Psi; \Delta \vdash t \colon A]\!]\,(g) &\in \mathcal{C}([\![\Delta]\!], [\![A]\!]) \\
[\![\Psi; \emptyset \vdash * \colon I]\!]\,(g) &= \mathrm{id}_{[\![I]\!]} \\
[\![\Psi; x \colon A \vdash x \colon A]\!]\,(g) &= \mathrm{id}_{[\![A]\!]} \\
[\![\Psi; \Delta \vdash \mathtt{inj}_\ell\, t \colon A \oplus B]\!]\,(g) &= \iota_l \circ [\![\Psi; \Delta \vdash t \colon A]\!]\,(g) \\
[\![\Psi; \Delta \vdash \mathtt{inj}_r\, t \colon A \oplus B]\!]\,(g) &= \iota_r \circ [\![\Psi; \Delta \vdash t \colon B]\!]\,(g) \\
[\![\Psi; \Delta_1, \Delta_2 \vdash t_1 \otimes t_2 \colon A \otimes B]\!]\,(g) &= [\![\Psi; \Delta_1 \vdash t_1 \colon A]\!]\,(g) \otimes [\![\Psi; \Delta_2 \vdash t_2 \colon B]\!]\,(g) \\
[\![\Psi; \Delta \vdash \mathtt{fold}\, t \colon \mu X.A]\!]\,(g) &= \alpha^{[\![X \vdash A]\!]} \circ [\![\Psi; \Delta \vdash t \colon A[\mu X.A/X]]\!]\,(g)
\end{aligned}
$$

$$
\begin{aligned}
[\![\Psi \vdash_\omega \omega \colon T]\!] &\in \mathbf{DCPO}([\![\Psi]\!], [\![T]\!]) \\
[\![\Psi, \phi \colon T \vdash_\omega \phi \colon T]\!] &= \pi_{[\![T]\!]} \\
[\![\Psi \vdash_\omega \omega_2 \omega_1 \colon T_2]\!] &= \mathrm{eval} \circ \langle [\![\Psi \vdash_\omega \omega_2 \colon T_1 \to T_2]\!], [\![\Psi \vdash_\omega \omega_1 \colon T_1]\!] \rangle \\
[\![\Psi \vdash_\omega \lambda\phi.\omega \colon T_1 \to T_2]\!] &= \mathrm{curry}([\![\Psi, \phi \colon T_1 \vdash_\omega \omega \colon T_2]\!]) \\
[\![\Psi \vdash_\omega \mathtt{fix}\ \phi.\omega \colon T]\!] &= \mathtt{fix}\ ([\![\Psi, \phi \colon T \vdash_\omega \omega \colon T]\!])
\end{aligned}
$$

▶ **Lemma 40.** *Given a dcpo Ξ, two objects X and Y of $\mathcal{C}$, a set of indices I and a family of Scott continuous maps $\xi_i \colon \Xi \to \mathcal{C}(X,Y)$ that are pairwise inverse compatible, the function $\bigvee_{i\in I} \xi_i : \Xi \to \mathcal{C}(X,Y)$ defined by $x \mapsto \bigvee_{i\in I} \xi_i(x)$ is Scott continuous.* ⌟

The interpretation of an iso abstraction is then given by:

$$[\![\Psi \vdash_\omega \{v_i \leftrightarrow e_i\}_{i\in I} \colon A \leftrightarrow B]\!] = \bigvee_{i\in I} (\mathrm{comp} \circ \langle [\![\Psi; \Delta_i \vdash e_i \colon B]\!], [\![\Psi; \Delta_i \vdash v_i \colon A]\!]^\circ \rangle)$$

The semantics is well-defined, in the sense that the interpretation of $\Psi \vdash_\omega \{v_i \leftrightarrow e_i\}_{i\in I} \colon A \leftrightarrow B$ is a Scott continuous map between the dcpos $[\![\Psi]\!]$ and $\mathcal{C}([\![A]\!], [\![B]\!])$.

6 Adequacy

We show a strong relationship between the operational semantics and the denotational semantics of the language. First, we fix a mathematical interpretation $[\![-]\!]$ in a join inverse rig category $\mathcal{C}$, that is **DCPO**-enriched and whose objects 0 and 1 are distinct.

Since the language handles non-termination, our adequacy statement links the denotational semantics to the notion of termination in the operational semantics: Given $\vdash t \colon A$, t is said to be *terminating* if there exists a value v such that $t \to^* v$. We either write $t\downarrow$, or $t \downarrow v$.

▶ **Theorem 41** (Adequacy). *Given $\vdash t \colon A$, $t\downarrow$ iff $[\![\vdash t \colon A]\!] \neq 0_{[\![A]\!]}$.*

Soundness. We start by showing the simple implication in Theorem 41 amount to soundness: the denotational semantics is stable w.r.t. computation.

▶ **Proposition 42** (Soundness). *Given a valid term judgement $\vdash t \colon A$, provided that $t \to t'$, then we have $[\![\vdash t \colon A]\!] = [\![\vdash t' \colon A]\!]$.* ⌟

We can conclude that if $\vdash t \colon A$ with $t\downarrow$, we have $[\![\vdash t \colon A]\!] \neq 0_{[\![A]\!]}$. This shows one of the implications in Theorem 41. For the proof of the other implication, we follow a syntactic approach, inspired by the proof in [35].

Proof of Adequacy. Our proof of adequacy involves a finitary sublanguage, where the number of recursive calls is controlled syntactically: instead of general fixpoints, we introduce a family of finitary fixpoints $\texttt{fix}^n\ \phi.\omega$, unfolding n times before reducing to the empty iso $\{\}$, corresponding to the diverging iso.

We show the adequacy result for the finitary terms thanks to strong normalisation, and then show that it implies adequacy for the whole language; this is achieved by observing that a normalising finitary term is also normalising in its non-finitary form.

7 Semantics preservation

In this section, we fix the interpretation $[\![-]\!]$ of the language in **PInj**, the category of sets and partial injections. This choice comes without any loss of generality (see [27]), and allows us to consider *computable* functions. In this section, we show that given a computable, reversible function $f\colon [\![A]\!] \to [\![B]\!]$, there exists an iso $\omega\colon A \leftrightarrow B$ such that $[\![\omega]\!] = f$. In order to do that, we fix a canonical flat representation of our types.

7.1 A Canonical Representation

We define a canonical representation of closed values of some type A into a new type $\mathrm{Enc} = \mathbb{B} \oplus \mathbb{1} \oplus \mathbb{1} \oplus \mathbb{1} \oplus \mathbb{1} \oplus \texttt{nat}$ (recall that $\mathbb{B} = \mathbb{1} \oplus \mathbb{1}$ and $\texttt{nat} = \mu X.\mathbb{1} \oplus X$). For simplicity let us name each the following terms of type Enc : $\mathbf{tt} = \texttt{inj}_\ell\ (\texttt{inj}_\ell\ ())$, $\mathbf{ff} = \texttt{inj}_\ell\ (\texttt{inj}_r\ ())$, $S = \texttt{inj}_r\ (\texttt{inj}_\ell\ ())$, $D^\oplus = \texttt{inj}_r\ (\texttt{inj}_r\ (\texttt{inj}_\ell\ ()))$, $D^\otimes = \texttt{inj}_r\ (\texttt{inj}_r\ (\texttt{inj}_r\ (\texttt{inj}_\ell\ ())))$, $D^\mu = \texttt{inj}_r\ (\texttt{inj}_r\ (\texttt{inj}_r\ (\texttt{inj}_r\ (\texttt{inj}_\ell\ ()))))$, and for every natural number n, we write $\tilde{n}$ for the term $\texttt{inj}_r\ (\texttt{inj}_r\ (\texttt{inj}_r\ (\texttt{inj}_r\ (\texttt{inj}_r\ (\texttt{inj}_r\ (\overline{n}))))))$, where $\overline{n}$ is the encoding of natural numbers, as given in Example 8. Now, given some closed type A, we can define $\lfloor - \rfloor_A\colon A \leftrightarrow [\mathrm{Enc}]$ the iso that transform any close value of type A into a list of Enc. The iso is defined inductively over A: $\lfloor - \rfloor_\mathbb{1} = \{() \leftrightarrow [S]\}$, and

$$\lfloor - \rfloor_{A\oplus B} = \left\{ \begin{array}{lcl} \texttt{inj}_\ell\ (x) & \leftrightarrow & \texttt{let}\, y = \lfloor x \rfloor_A\ \texttt{in}\ D^\oplus :: \mathbf{ff} :: y \\ \texttt{inj}_r\ (x) & \leftrightarrow & \texttt{let}\, y = \lfloor x \rfloor_B\ \texttt{in}\ D^\oplus :: \mathbf{tt} :: y \end{array} \right\},$$

$$\lfloor - \rfloor_{A\otimes B} = \left\{ \begin{array}{lcl} \langle x, y \rangle & \leftrightarrow & \texttt{let}\, x' = \lfloor x \rfloor_A\ \texttt{in let}\, y' = \lfloor y \rfloor_B\ \texttt{in} \\ & & \texttt{let}\, \langle z, n \rangle = ++\ \langle x', y' \rangle\ \texttt{in}\ D^\otimes :: \tilde{n} :: z \end{array} \right\},$$

$$\lfloor - \rfloor_{\mu X.A} = \left\{ \texttt{fold}\, x \quad \leftrightarrow \quad \texttt{let}\, y = \lfloor x \rfloor_{A[\mu X.A/X]}\ \texttt{in}\ D^\mu :: y \right\},$$

where the iso $++\colon [A] \otimes [A] \leftrightarrow [A] \otimes \texttt{nat}$ which concatenate two lists is defined as:

$$\texttt{fix}\ f. \left\{ \begin{array}{lcl} \langle [], x \rangle & \leftrightarrow & \langle x, 0 \rangle \\ \langle h :: t, x \rangle & \leftrightarrow & \texttt{let}\, \langle y, n \rangle = f\ \langle t, x \rangle\ \texttt{in}\ \langle h :: y, S(n) \rangle \end{array} \right\}.$$

7.2 Capturing every computable injection

With this encoding, every iso $\omega\colon A \leftrightarrow B$ can be turned into another iso $\lfloor \omega \rfloor\colon [\mathrm{Enc}] \leftrightarrow [\mathrm{Enc}]$ by composing $\lfloor - \rfloor_A$, followed by ω, followed by $\lfloor - \rfloor_B^{-1}$. This is in particular the case for isos that are the images of a Turing Machine. We are now ready to see how every computable function f from $[\![A]\!] \to [\![B]\!]$ can be turned into an iso whose semantics is f. Given a computable function $f\colon [\![A]\!] \to [\![B]\!]$, call M_f the RTM computing f. Since f is in **PInj**, its output uniquely determines its input. Following [4], given the output of the machine M_f there exists another Turing Machine M_f' which takes this output and recover the original input of M_f. In our encoding of a RTM, the iso will have another additional garbage which consist of a natural

number, i.e. the number of steps of the RTM M_f. Using $\texttt{GarbRem}(\text{isos}(M_f), \text{isos}(M'_f))$ we can obtain a single iso, from the encoding of A to the encoding of B, without any garbage left. This also ensures that $\left[\!\left[\texttt{GarbRem}(\text{isos}(M_f), \text{isos}(M'_f))\right]\!\right](x) = (\llbracket \text{isos}(M_f) \rrbracket (x))_1$, for any input x.

▶ **Theorem 43** (Computable function as Iso). *Given a computable function* $f \colon \llbracket A \rrbracket \to \llbracket B \rrbracket$*, let* $g \colon \llbracket [\text{Enc}] \otimes [\text{Enc}] \rrbracket \to \llbracket [\text{Enc}] \otimes [\text{Enc}] \rrbracket$ *be defined as* $g = \llbracket \lfloor - \rfloor_B \rrbracket \circ f \circ \llbracket \lfloor - \rfloor_A^{-1} \rrbracket$*, and let* $\omega \colon A \leftrightarrow B$ *be defined as* $\{x \leftrightarrow \texttt{let}\, y = \lfloor x \rfloor_A \,\texttt{in let}\, y' = \texttt{GarbRem}(\text{isos}(M_g), \text{isos}(M'_g))\ y \,\texttt{in let}\, z = \lfloor y' \rfloor_B^{-1} \,\texttt{in}\ z\}$*. Then* $\llbracket \omega \rrbracket = f$*.* ⌟

8 Conclusion

In this paper, we built upon the language presented in [7, 8, 36] in order to represent any partial injective function which can manipulate inductive types. We showed how one can encode any Reversible Turing Machine, hence the (reversible) Turing Completeness, and we gave a denotational semantics based on join inverse rig categories, together with a soundness and adequacy theorem. Most notably, we showed that for any computable function f from **PInj**, there exists an iso whose semantics is f, thus our language fully characterises all of the computable morphisms in **PInj**.

References

1 Holger Bock Axelsen and Robert Glück. A simple and efficient universal reversible turing machine. In Adrian-Horia Dediu, Shunsuke Inenaga, and Carlos Martín-Vide, editors, *Language and Automata Theory and Applications*, pages 117–128, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. doi:10.1007/978-3-642-21254-3_8.

2 Holger Bock Axelsen and Robin Kaarsgaard. Join inverse categories as models of reversible recursion. In Bart Jacobs and Christof Löding, editors, *Proceedings of the 19th International Conference on Foundations of Software Science and Computation Structures (FOSSACS'16)*, volume 9634 of *Lecture Notes in Computer Science*, pages 73–90, Eindhoven, The Netherlands, 2016. Springer. doi:10.1007/978-3-662-49630-5_5.

3 Henk Barendregt. 'the lambda calculus: its syntax and semantics'. *Studies in logic and the foundations of Mathematics*, 1984.

4 Charles H Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*, 17(6):525–532, 1973. doi:10.1147/rd.176.0525.

5 Antoine Bérut, Artak Arakelyan, Artyom Petrosyan, Sergio Ciliberto, Raoul Dillenschneider, and Eric Lutz. Experimental verification of landauer's principle linking information and thermodynamics. *Nature*, 483(7388):187–189, 2012.

6 Kostia Chardonnet. *Towards a Curry-Howard Correspondence for Quantum Computation*. Theses, Université Paris-Saclay, January 2023. URL: https://theses.hal.science/tel-03959403.

7 Kostia Chardonnet, Louis Lemonnier, and Benoît Valiron. Categorical semantics of reversible pattern-matching. *Electronic Proceedings in Theoretical Computer Science*, 351:18–33, December 2021. doi:10.4204/eptcs.351.2.

8 Kostia Chardonnet, Alexis Saurin, and Benoît Valiron. A Curry-Howard correspondence for linear, reversible computation. In Bartek Klin and Elaine Pimentel, editors, *31st EACSL Annual Conference on Computer Science Logic (CSL 2023)*, volume 252 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:18, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CSL.2023.13.

9 J. Robin B. Cockett and Stephen Lack. Restriction categories I: Categories of partial maps. *Theoretical Computer Science*, 270(1):223–259, 2002. doi:10.1016/S0304-3975(00)00382-0.

10 J. Robin B. Cockett and Stephen Lack. Restriction categories ii: partial map classification. *Theoretical Computer Science*, 294(1):61–102, 2003. doi:10.1016/S0304-3975(01)00245-6.

11 Robin Cockett and Stephen Lack. Restriction categories III: Colimits, partial limits and extensivity. *Mathematical Structures in Computer Science*, 17(4):775–817, 2007. doi:10.1017/S0960129507006056.

12 M.P. Fiore. *Axiomatic Domain Theory in Categories of Partial Maps*. Distinguished Dissertations in Computer Science. Cambridge University Press, 2004. URL: https://books.google.co.uk/books?id=rsIAmbc2cIoC.

13 Brett Gordon Giles. *An Investigation of Some Theoretical Aspects of Reversible Computing*. PhD thesis, University of Calgary, 2014. doi:10.11575/PRISM/24917.

14 Robert Glück and Robin Kaarsgaard. A categorical foundation for structured reversible flowchart languages: Soundness and adequacy. *Log. Methods Comput. Sci.*, 14(3), 2018. doi:10.23638/LMCS-14(3:16)2018.

15 Robert Glück, Robin Kaarsgaard, and Tetsuo Yokoyama. Reversible programs have reversible semantics. In Emil Sekerinski, Nelma Moreira, José N. Oliveira, Daniel Ratiu, Riccardo Guidotti, Marie Farrell, Matt Luckcuck, Diego Marmsoler, José Campos, Troy Astarte, Laure Gonnord, Antonio Cerone, Luis Couto, Brijesh Dongol, Martin Kutrib, Pedro Monteiro, and David Delmas, editors, *Formal Methods. FM 2019 International Workshops - Porto, Portugal, October 7-11, 2019, Revised Selected Papers, Part II*, volume 12233 of *Lecture Notes in Computer Science*, pages 413–427. Springer, 2019. doi:10.1007/978-3-030-54997-8_26.

16 Xiuzhan Guo. *Products, Joins, Meets, and Ranges in Restriction Categories*. PhD thesis, University of Calgary, 2012. doi:10.11575/PRISM/4745.

17 Chris Heunen, Robin Kaarsgaard, and Martti Karvonen. Reversible effects as inverse arrows. In Sam Staton, editor, *Proceedings of the 34th Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXIV)*, volume 341 of *Electronic Notes in Theoretical Computer Science*, pages 179–199, Dalhousie University, Halifax, Canada, 2018. Elsevier. doi:10.1016/j.entcs.2018.11.009.

18 Chris Heunen and Martti Karvonen. Reversible monadic computing. In Dan Ghica, editor, *Proceedings of the 31st Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXI)*, volume 319 of *Electronic Notes in Theoretical Computer Science*, pages 217–237, Nijmegen, The Netherlands, 2015. doi:10.1016/j.entcs.2015.12.014.

19 Petur Andrias Højgaard Jacobsen, Robin Kaarsgaard, and Michael Kirkedal Thomsen. CoreFun: A typed functional reversible core language. In Jarkko Kari and Irek Ulidowski, editors, *Reversible Computation - 10th International Conference, RC 2018, Leicester, UK, September 12-14, 2018, Proceedings*, volume 11106 of *Lecture Notes in Computer Science*, pages 304–321. Springer, 2018. doi:10.1007/978-3-319-99498-7_21.

20 Rosham P. James and Amr Sabry. Theseus: A high-level language for reversible computing. Draft, available at https://legacy.cs.indiana.edu/~sabry/papers/theseus.pdf, 2014.

21 Xiaodong Jia, Bert Lindenhovius, Michael Mislove, and Vladimir Zamdzhiev. Commutative monads for probabilistic programming languages. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, June 2021. doi:10.1109/lics52264.2021.9470611.

22 Robin Kaarsgaard. Condition/decision duality and the internal logic of extensive restriction categories. In Barbara König, editor, *Proceedings of the 35th Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXV)*, volume 347 of *Electronic Notes in Theoretical Computer Science*, pages 179–202, London, UK, 2019. doi:10.1016/j.entcs.2019.09.010.

23 Robin Kaarsgaard. Inversion, iteration, and the art of dual wielding. In Michael Kirkedal Thomsen and Mathias Soeken, editors, *Proceedings of the 11th International Conference on Reversible Computation (RC 2019)*, volume 11497 of *Lecture Notes in Computer Science*, pages 34–50, Lausanne, Switzerland, 2019. Springer. doi:10.1007/978-3-030-21500-2_3.

24 Robin Kaarsgaard, Holger Bock Axelsen, and Robert Glück. Join inverse categories and reversible recursion. *Journal of Logical and Algebraic Methods in Programming*, 87:33–50, 2017. doi:10.1016/j.jlamp.2016.08.003.

25 Robin Kaarsgaard and Mathys Rennela. Join inverse rig categories for reversible functional programming, and beyond. In Ana Sokolova, editor, Proceedings 37th Conference on *Mathematical Foundations of Programming Semantics,* Hybrid: Salzburg, Austria and Online, 30th August - 2nd September, 2021, volume 351 of *Electronic Proceedings in Theoretical Computer Science*, pages 152–167. Open Publishing Association, 2021. doi:10.4204/EPTCS.351.10.

26 Robin Kaarsgaard and Niccolò Veltri. En garde! unguarded iteration for reversible computation in the delay monad. In Graham Hutton, editor, *Proceedings of the 13th International Conference on Mathematics of Program Construction (MPC 2019)*, volume 11825 of *Lecture Notes in Computer Science*, pages 366–384, Porto, Portugal, October 2019. Springer Verlag. doi:10.1007/978-3-030-33636-3_13.

27 J. Kastl. Inverse categories. In *Algebraische Modelle, Kategorien und Gruppoide*, Studien zur Algebra und ihre Anwendungen, Band 7, pages 51–60. Berlin, Akademie-Verlag, 1979.

28 G.M Kelly. Tensor products in categories. *Journal of Algebra*, 2(1):15–37, 1965. doi:10.1016/0021-8693(65)90022-0.

29 Max Kelly. *Basic concepts of enriched category theory*, volume 64. CUP Archive, 1982.

30 Rolf Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development.*, 5(3):183–191, 1961. doi:10.1147/rd.53.0183.

31 Christopher Lutz. Janus: a time-reversible language. Letter to Rolf Landauer, posted online by Tetsuo Yokoyama on http://www.tetsuo.jp/ref/janus.html, 1986.

32 Ian Mackie. The geometry of interaction machine. In *Proceedings of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL'95*, pages 198–208. ACM Press, 1995. doi:10.1145/199448.199483.

33 J.-M. Maranda. Formal categories. *Canadian Journal of Mathematics*, 17:758–801, 1965. doi:10.4153/CJM-1965-076-0.

34 Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information.* Cambridge University Press, 2002.

35 Michele Pagani, Peter Selinger, and Benoît Valiron. Applying quantitative semantics to higher-order quantum computing. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '14, pages 647–658, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2535838.2535879.

36 Amr Sabry, Benoît Valiron, and Juliana Kaizer Vizzotto. From symmetric pattern-matching to quantum control. In Christel Baier and Ugo Dal Lago, editors, *Proceedings of the 21st International Conference on Foundations of Software Science and Computation Structures (FOSSACS'18)*, volume 10803 of *Lecture Notes in Computer Science*, pages 348–364, Thessaloniki, Greece, 2018. Springer. doi:10.1007/978-3-319-89366-2_19.

37 Michael Kirkedal Thomsen and Holger Bock Axelsen. Interpretation and programming of the reversible functional language RFUN. In Ralf Lämmel, editor, *Proceedings of the 27th Symposium on the Implementation and Application of Functional Programming Languages, IFL 2015, Koblenz, Germany, September 14-16, 2015*, pages 8:1–8:13. ACM, 2015. doi:10.1145/2897336.2897345.

38 Tetsuo Yokoyama, Holger Bock Axelsen, and Robert Glück. Towards a reversible functional language. In Alexis De Vos and Robert Wille, editors, *Revised Papers of the Third International Workshop on Reversible Computation (RC'11)*, volume 7165 of *Lecture Notes in Computer Science*, pages 14–29, Gent, Belgium, 2012. Springer. doi:10.1007/978-3-642-29517-1_2.

39 Tetsuo Yokoyama, Holger Bock Axelsen, and Robert Glück. Fundamentals of reversible flowchart languages. *Theoretical Computer Science*, 611:87–115, 2016. doi:10.1016/j.tcs.2015.07.046.

40 Tetsuo Yokoyama and Robert Glück. A reversible programming language and its invertible self-interpreter. In G. Ramalingam and Eelco Visser, editors, *Proceedings of the 2007 ACM SIGPLAN Workshop on Partial Evaluation and Semantics-based Program Manipulation, PEPM 2007, Nice, France, January 15-16, 2007*, pages 144–153, 2007. doi:10.1145/1244381.1244404.

On Iteration in Discrete Probabilistic Programming

Mateo Torres-Ruiz ✉
University College London, United Kingdom

Robin Piedeleu
University College London, United Kingdom

Alexandra Silva
Cornell University, Ithaca, NY, United States of America

Fabio Zanasi
University College London, United Kingdom
University of Bologna, OLAS team (INRIA), Italy

Abstract

Discrete probabilistic programming languages provide an expressive tool for representing and reasoning about probabilistic models. These languages typically define the semantics of a program through its posterior distribution, obtained through exact inference techniques.

While the semantics of standard programming constructs in this context is well understood, there is a gap in extending these languages with tools to reason about the asymptotic behaviour of programs. In this paper, we introduce unbounded iteration in the context of a discrete probabilistic programming language, give it a semantics, and show how to compute it exactly. This allows us to express the stationary distribution of a probabilistic function while preserving the efficiency of exact inference techniques. We discuss the advantages and limitations of our approach, showcasing their practical utility by considering examples where bounded iteration poses a challenge due to the inherent difficulty of assessing the proximity of a distribution to its stationary point.

2012 ACM Subject Classification Theory of computation → Program semantics

Keywords and phrases Probabilistic programming, Programming languages semantics, Unbounded iteration

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.20

Funding This work was partially supported by ERC grant Autoprobe (no. 101002697; Piedeleu, Silva, and Torres-Ruiz).
Fabio Zanasi: Fabio Zanasi acknowledges support from EPSRC EP/V002376/1 and MIUR P2022HXNSC (PRIN 2022 PNRR - Next Generation EU).

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

1 Introduction

Probabilistic Programming Languages (PPLs) combine the expressive power of classical programming constructs with the ability to draw random values from probability distributions and condition on the variables declared [18]. Unlike traditional programming languages, where evaluation leads to executing a routine, evaluating a probabilistic program results in computing the posterior probability distribution it specifies, a task called *inference*.

Inference can be either approximate [9, 17, 44], typically performed via sampling, or exact [7, 16, 35, 37], where the aim is to compute the *exact* posterior specified by the program. While most PPLs emphasise the use of continuous random variables, discrete random variables lend themselves to exact inference techniques. Moreover, they are better suited to several application domains, from cryptography [1, 33] to networks [14, 15, 42] or graphs [39]. These appealing properties have sparked renewed interest in discrete PPLs.

While finiteness of the discrete setting permits the computation of an exact posterior, exact inference is known to be undecidable for programs with an unbounded domain and #P-complete for programs which make use of Boolean variables only [40]. To tackle this difficulty, PPL compilers use data structures that exploit independence between variables, leading to efficient and scalable exact inference [24, 7].

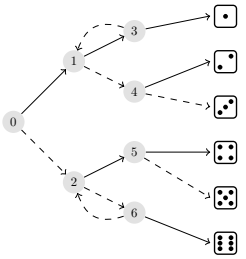
Even though the semantics of probabilistic languages have been extensively studied ever since the seminal work of Kozen [29], and the standard programming constructs are well-understood for discrete distributions, their extension to *unbounded iteration in the presence of exact inference* has been mostly absent. Possibly, due to the fact that unbounded iteration is seen to introduce non-termination, which can take the language semantics and inference capabilities outside the domain of discrete probabilities. As we show in this paper, this does not have to be the case: we extend a discrete PPL with unbounded iteration, while retaining the ability to perform exact inference.

Unbounded iteration is important to model probability distributions arising as the asymptotic behaviour of a (discrete-time) stochastic process, which are the natural object of study in a multitude of applications: given a ciphertext, we can construct a Markov chain to represent the posterior distribution of encryption keys [12]; in molecular physics, protein trajectories can be modeled using discrete-time Markov Chains (DTMCs), providing insights into the protein's biological function through their stationary distribution [38]; random walks on the symmetric group offer various applications such as diffusion models or shuffling schemes, where stationarity ensures reaching some optimal state [31]. From a probabilistic perspective, these can be seen as the outcome of iterating a program which models the one-step dynamics of the chain. Of course, one can always iterate such process a fixed number of times or until some condition is met. As we will see, this is often insufficient, and does not lead to the desired asymptotic behaviour in general. To reach the asymptotic limit, dedicated programming constructs for unbounded iteration are required.

The language we chose to extend is Dice [24], a functional PPL designed to handle discrete random variables and perform exact inference. Dice has a mature implementation and a very efficient exact inference engine. Before delving into our contributions, we provide a simple example that further motivates the use of unbounded iteration in probabilistic programming.

▶ **Example 1.** Consider the following probabilistic model which simulates a die using repeated throws of a coin c [28]. Starting at the root vertex, s_0, the model iteratively flips c and branches with a solid line whenever it lands heads, and with a dashed line when it results in tails. The coin-flipping procedure continues until a leaf is reached, i.e., until the value of the die is determined. We can write this as a probabilistic program:

```
fun knuth_yao(s) {
  if s == 0 then (if flip 1/2 then 1 else 2) else
  if s == 1 then (if flip 1/2 then 3 else 4) else
  if s == 2 then (if flip 1/2 then 5 else 6) else
  if s == 3 then (if flip 1/2 then ⚀ else 1) else
  if s == 4 then (if flip 1/2 then ⚁ else ⚂) else
  if s == 5 then (if flip 1/2 then ⚃ else ⚄) else
  if s == 6 then (if flip 1/2 then ⚅ else 2) else s }
```



The proposed model has 13 possible states: $\{0, \ldots, 6, ⚀, \ldots, ⚅\}$. The transition from the current state s is dependent upon both s and the result of a fair coin flip. For instance, starting from 0, we transition to either 1 or 2 based on the outcome of a coin flip, and it takes at least three consecutive runs to reach a $⚀, \ldots, ⚅$ state. Each function call yields not a single state but a *probability distribution* on the 13 potential states. While careful analysis

of this and similar programs make it clear that *in the limit* the model indeed simulates a six-side fair die [26, 27], it is not immediately apparent that by iteratively applying this function, we will attain each possible die outcome with a probability of $\frac{1}{6}$.

Contributions. We extend Dice [24], a discrete PPL, with an unbounded iteration operator, give it a semantics that remains within the discrete realm, and show how to compute it exactly. Furthermore, we also consider an alternative form of iteration, which is more closely related to probabilistic while-loops and whose semantics is given by a least fixpoint.

We begin in Section 2 by formalizing the semantics of Dice. In Section 3 we highlight the difficulties encountered when attempting to use bounded iteration to obtain the asymptotic behaviour of some stochastic process. We then introduce the syntax and semantics for our proposed unbounded iteration operator in Section 4. An alternative form of iteration is then presented in Section 5, followed by a discussion of the implementation of both iterators, and an empirical evaluation of their performance in Section 6. Finally, we conclude with a discussion of related work and a few suggestions for future research in Section 7.

2 Dice: An Efficient Discrete PPL

In this section we review the syntax and semantics of Dice [24], a first-order non-recursive functional language featuring the essential characteristics of a discrete PPL. It supports Bernoulli distributions and has first-class observations which allow for Bayesian inference, while it offers an efficient approach to probabilistic inference by exploiting the internal representation of its programs.

2.1 Syntax

Dice supports Boolean and product types, as well as common branching and customary Boolean operators. The language features two probabilistic constructs: one for defining Bernoulli distributions (flip) and one for Bayesian evidence (observe). The first of these takes a real number as a parameter $\theta \in [0, 1]$ and denotes a distribution that evaluates to true with probability θ and false with probability $1 - \theta$, while the second takes an atomic expression a as a parameter and incorporates it as evidence that a evaluates to true throughout the program. This effectively changes the probability of all program paths where a does not hold to 0. The type of a Dice program is a product of Booleans, determined by the rules in Fig. 1.

A Dice program consists of a sequence of non-recursive functions followed by a main expression. We will later see that these expressions can be interpreted as discrete distributions on the declared values. Furthermore, it supports syntactic sugar for logical operations $(\wedge, \vee, \neg)$, bounded loops, bounded-size integers, and discrete distributions over k integers.

▶ **Example 2.** Consider the following straightforward setting: we have observed that trains to the nearest city experience delays with a 0.7 chance when it is raining and 0.3 otherwise. On any given day, there is a 0.1 probability of rain. If we observe that the trains are running late, what is the probability that it is raining? We can write down this in Dice as follows:

```
let x = flip 0.1 in
let y = if x then flip 0.7 else flip 0.3 in
let z = observe(y) in x
```

2.2 Semantics

Recall that a function $p : X \to [0,1]$ is a *probability distribution* whenever $\sum_{x \in X} p(x) = 1$, and is called a *subprobability distribution* when $\sum_{x \in X} p(x) < 1$. Closed Dice expressions can be viewed as a means of representing a subprobability distribution over the set of values attainable by the variables declared. The distributions expressible in Dice are furthermore *finitely supported*. That is, the set $supp(p) := \{x \in X \mid p(x) > 0\}$ is finite.

We can define well-typed expressions $\Gamma \vdash e \colon \tau$ and these will represent a (sub)probability distribution over the tuple of variables τ conditional on the value of the free variables that appear in Γ. To give a compositional interpretation of Dice expressions, we will interpret conditional distributions in a category of certain linear maps represented as matrices. In this sense, our presentation of the semantics of Dice is superficially different from that of the original paper, though the two are equivalent.

A nonnegative matrix is *stochastic* if each column sums to 1 and *substochastic* if the total of each column falls between 0 and 1. We write $\mathbb{R}^X$ for the vector space of real-valued functions on $\mathbb{R}$ which we equip with its Dirac basis $\{\delta_x \,|\, x \in X\}$ where $\delta_x(y) = 1$ if $x = y$ and 0 otherwise. Conditional (sub)distributions $p(y|x)$ can be thought of as linear maps $\varphi_p : \mathbb{R}^X \to \mathbb{R}^Y$ where φ_p is given by extending the mapping $x \mapsto \sum_y p(y|x)\delta_y$ linearly. The map φ_p then has the additional property that its matrix representation in the Dirac bases for $\mathbb{R}^X$ and $\mathbb{R}^Y$ is (sub)stochastic. We also call such maps (sub)stochastic maps. More generally, in what follows, we will speak of linear maps and matrices interchangeably, since all of our vector spaces are equipped with a chosen basis. We will use $\circ$ to denote composition of linear maps or simply juxtaposition to denote the product of matrices.

Since the product of substochastic matrices is substochastic, substochastic maps form a category Stoch with finite sets as objects and morphisms $X \to Y$ substochastic maps $\mathbb{R}^X \to \mathbb{R}^Y$. This is the category in which we interpret Dice programs. To interpret tuples, we also need to equip Stoch with the monoidal product given by the Cartesian product of sets on objects, and the usual tensor product on linear maps or, alternatively, the Kronecker product $\otimes$ on matrices. Note that the set of Diracs $\{\delta_x \otimes \delta_y \,|\, (x,y) \in X \times Y\}$ forms a basis of $\mathbb{R}^{X \times Y} \cong \mathbb{R}^X \otimes \mathbb{R}^Y$. This defines a monoidal structure over the category Stoch, which is moreover symmetric monoidal by equipping it with $\mathsf{swap}_{X,Y} : X \times Y \to Y \times X$ given by extending the mapping $\delta_x \otimes \delta_y \mapsto \delta_y \otimes \delta_x$ linearly.

▶ Remark 3. For the categorically-minded reader, the interpretation of Dice could also be given in the Kleisli category of the (sub)probability monad, which is equivalent to the category of (sub)stochastic linear maps. Therefore, most of the subsequent development could be phrased in this more categorical setting. We prefer a straightforward linear algebraic presentation, because our treatment of iteration in the coming sections makes use of standard results about Markov chains, which are usually formulated in a less abstract setting.

The type of the Booleans can be interpreted in Stoch as $[\![\mathbb{B}]\!] = \mathbb{B} = \{\mathsf{true}, \mathsf{false}\}$ and extends $[\![\cdot]\!]$ to arbitrary types and contexts $[\![()]\!] = 1 \cong \{0\}$, $[\![\tau \times \tau']\!] = [\![\tau]\!] \times [\![\tau']\!]$, $[\![\tau, \ldots, \tau']\!] = [\![\tau]\!] \times \cdots \times [\![\tau']\!]$. To define the semantics of open terms, we will need the following stochastic maps for variable management.

- To discard a variable we will use the linear map dis_X obtained by extending $\delta_x \mapsto 1$ for all $x \in X$ linearly; for a single Boolean variable $\mathsf{dis}_{\mathbb{B}}$ is given by the stochastic matrix/row vector $\begin{pmatrix} 1 & 1 \end{pmatrix}$.
- To use a variable more than once, we use the linear map copy_X obtained by extending $\delta_x \mapsto \delta_x \otimes \delta_x$ linearly, for each $x \in X$; for a single Boolean variable $\mathsf{copy}_{\mathbb{B}}$ is represented by the stochastic matrix $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}^\top$.

$$\frac{}{\Gamma \vdash \texttt{true} : \mathbb{B}}\ (\text{true}) \qquad \frac{}{\Gamma \vdash \texttt{false} : \mathbb{B}}\ (\text{false}) \qquad \frac{}{\Gamma \vdash \texttt{flip}\ \theta : \mathbb{B}}\ (\text{flip})$$

$$\frac{\Gamma \vdash x : \mathbb{B}}{\Gamma \vdash \texttt{observe}\ x : \mathbb{B}}\ (\text{obs}) \qquad \frac{}{\Gamma, x : \tau, \Gamma' \vdash x : \tau}\ (\text{var}) \qquad \frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2}\ (\text{tup})$$

$$\frac{\Gamma \vdash e : \tau_l \times \tau_r}{\Gamma \vdash \texttt{fst}\ e : \tau_l}\ (\pi_1) \qquad \frac{\Gamma \vdash e : \tau_l \times \tau_r}{\Gamma \vdash \texttt{snd}\ e : \tau_r}\ (\pi_2) \qquad \frac{\Gamma \vdash e_0 : \tau_0 \qquad \Gamma, x : \tau_0 \vdash e_1 : \tau}{\Gamma \vdash \texttt{let}\ x = e_0\ \texttt{in}\ e_1 : \tau}\ (\text{let})$$

$$\frac{\Gamma \vdash g : \mathbb{B} \qquad \Gamma \vdash e_T : \tau \qquad \Gamma \vdash e_F : \tau}{\Gamma \vdash \texttt{if}\ g\ \texttt{then}\ e_T\ \texttt{else}\ e_F : \tau}\ (\text{ite}) \qquad \frac{\Gamma, x \mapsto \tau \vdash e : \tau'}{\Gamma \vdash \texttt{fun}\ f(x : \tau) : \tau'\ \{e\} : \tau \to \tau'}\ (\text{fn})$$

$$\frac{\Gamma \vdash f : \tau \to \tau' \qquad \Gamma \vdash z : \tau}{\Gamma \vdash f(z) : \tau'}\ (\text{call})$$

Figure 1 Dice typing rules.

In general a typing statement $\Gamma \vdash t : \tau$ is interpreted as a substochastic map $[\![t]\!] : [\![\Gamma]\!] \to [\![\tau]\!]$, where the semantics $[\![\cdot]\!]$ of an open term $\Gamma \vdash t : \tau$ is given recursively on the structure of the typing rules of Fig. 1.

- By convention, the Booleans are represented as the following column vectors: $[\![\mathsf{true}]\!] = \delta_{\mathsf{true}} = \begin{pmatrix} 1 & 0 \end{pmatrix}^\top$ and $[\![\texttt{false}]\!] = \delta_{\mathsf{false}} = \begin{pmatrix} 0 & 1 \end{pmatrix}^\top$.
- $[\![\texttt{flip}\ \theta]\!]$ is given by the Bernoulli distribution that assigns true with probability θ and false with probability $1 - \theta$. This is interpreted as the map $[\![\Gamma]\!] \xrightarrow{\mathsf{dis}_{[\![\Gamma]\!]}} 1 \xrightarrow{[\theta\ 1-\theta]^T} [\![\mathbb{B}]\!]$. Notice that $[\![\texttt{false}]\!]$ and $[\![\texttt{true}]\!]$ can be obtained as $\texttt{flip}\ \theta$ (in the empty context) where θ takes values 0 and 1, respectively.

$$[\![\Gamma, \tau, \Gamma']\!] \cong [\![\Gamma]\!] \times [\![\tau]\!] \times [\![\Gamma']\!] \xrightarrow{\mathsf{dis}_{[\![\Gamma]\!]} \otimes \mathsf{id}_{[\![\tau]\!]} \otimes \mathsf{dis}_{[\![\Gamma']\!]}} 1 \times [\![\tau]\!] \times I \cong [\![\tau]\!]$$

- Variables are introduced through let-expressions, which are interpreted as the map

$$[\![\Gamma]\!] \xrightarrow{\mathsf{copy}_{[\![\Gamma]\!]}} [\![\Gamma]\!] \times [\![\Gamma]\!] \xrightarrow{[\![e_0]\!] \otimes \mathsf{id}_{[\![\Gamma]\!]}} [\![\tau_0]\!] \times [\![\Gamma]\!] \xrightarrow{[\![e_1]\!]} [\![\tau]\!]$$

- Tuples (e_1, e_2) are interpreted as a joint distribution over the expressions involved, given by the map $[\![\Gamma]\!] \xrightarrow{\mathsf{copy}_{[\![\Gamma]\!]}} [\![\Gamma]\!] \times [\![\Gamma]\!] \xrightarrow{[\![e_1]\!] \otimes [\![e_2]\!]} [\![\tau_1]\!] \times [\![\tau_2]\!] \cong [\![\tau_1 \times \tau_2]\!]$.
- The projections $[\![\texttt{fst}\ e]\!]$ and $[\![\texttt{snd}\ e]\!]$ correspond to the marginalisation of joint distributions, given by applying $\mathsf{dis}_{[\![\tau]\!]}$ to the component over which we wish to marginalise, *i.e.*, $[\![\texttt{fst}\ e]\!] = [\![\Gamma]\!] \xrightarrow{[\![e]\!]} [\![\tau_l]\!] \times [\![\tau_r]\!] \xrightarrow{\mathsf{id}_{[\![\tau_l]\!]} \otimes \mathsf{dis}_{[\![\tau_r]\!]}} [\![\tau_l]\!]$ and $[\![\texttt{snd}\ e]\!] = [\![\Gamma]\!] \xrightarrow{[\![e]\!]} [\![\tau_l]\!] \times [\![\tau_r]\!] \xrightarrow{\mathsf{dis}_{[\![\tau_l]\!]} \otimes \mathsf{id}_{[\![\tau_r]\!]}} [\![\tau_r]\!]$.
- Conditionals $[\![\texttt{if}\ g\ \texttt{then}\ e_T\ \texttt{else}\ e_F]\!]$ take an expression g as guard whose truth value determines which of the branchings will be selected; its semantics is given by the map

$$[\![\Gamma]\!] \xrightarrow{\mathsf{copy}_{[\![\Gamma]\!]}} [\![\Gamma]\!] \times [\![\Gamma]\!] \xrightarrow{\mathsf{id}_{[\![\Gamma]\!]} \otimes \mathsf{copy}_{[\![\Gamma]\!]}} [\![\Gamma]\!] \times ([\![\Gamma]\!] \times [\![\Gamma]\!]) \xrightarrow{[\![g]\!] \otimes ([\![e_F]\!] \otimes [\![e_T]\!])} \mathbb{B} \times ([\![\tau]\!] \times [\![\tau]\!]) \xrightarrow{\mathsf{join}_{[\![\tau]\!]}} [\![\tau]\!]$$

  where $\mathsf{join}_X : \mathbb{B} \times X \times X \to X$ is the linear map obtained by extending the following mapping linearly: $\mathsf{join}_X(\delta_{\mathsf{true}}, x, y) = x$ and $\mathsf{join}_X(\delta_{\mathsf{false}}, x, y) = y$. For simple Boolean expressions, *i.e.*, for $[\![\tau]\!] = \mathbb{B}$, the corresponding 2-by-8 matrix is $\begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}$.
- Observe statements $[\![\texttt{observe}\ x]\!]$ incorporate evidence by assigning a 0 probability to executions where x does not hold. The interpretation of observations is given by the map $\mathbb{B} \xrightarrow{\mathsf{observe}} \mathbb{B}$, defined as the 2-by-2 matrix $\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$. Note that this is the only *sub*stochastic map which is not stochastic of all the Dice constructs.

- Functions are simply interpreted by keeping an environment mapping names to the semantics of the corresponding term: the (fn) rule then creates a new name for a given function, and (call) is interpreted as function application (note that Dice does not allow higher-order functions). This aspect of the semantics is not of much relevance to the development below, so we refer the interested reader to the original paper on Dice [24].

▶ **Example 4.** The semantics of the program in Example 2 is $(\mathsf{id}_{\mathbb{B}} \otimes (\mathsf{dis}_{\mathbb{B}} \circ \mathsf{observe})) \circ (\mathsf{id}_{\mathbb{B}} \otimes \mathsf{join}_{\mathbb{B}}) \circ (\mathsf{copy}_{\mathbb{B}} \otimes \mathsf{id}_{\mathbb{B}} \otimes \mathsf{id}_{\mathbb{B}}) \circ (\llbracket \mathsf{flip}\ 0.1 \rrbracket \otimes \llbracket \mathsf{flip}\ 0.7 \rrbracket \otimes \llbracket \mathsf{flip}\ 0.3 \rrbracket)$ which computes to $\begin{pmatrix} 0.07 \\ 0.27 \end{pmatrix}$.

Notice that the use of observe in the program makes this an unnormalised subprobability distribution. To interpret the outcome of a computation, the user needs a bona fide probability distribution. Hence, the Dice compiler automatically normalises the result at the end of a computation. From a denotational perspective, there is not much difference between the two, since one can always renormalise a nonzero subprobability. Note however that normalising is an expensive operation in principle. For this reason, the Dice compiler keeps track of the subprobability distribution that a given term denotes *and* of its associated normalisation constant as well. See Appendix A for a discussion of Dice's implementation details.

3 Iteration in PPLs: Existing Approaches and Challenges

It is straightforward to extend a discrete PPL with an operation for *bounded* iteration. In Dice, this problem was already addressed in [24] through the introduction of bounded loops, $\texttt{iterate}(f, i, k)$, where f is a function name, i an initialization expression and k an integer specifying the number of times the function $f : \tau \to \tau$ should be iterated with a starting value i. While this allows for denoting a loop that terminates after a known number of iterations, it does not allow us to obtain the limiting behaviour of f. In this section, we review these limitations, thus justifying the need for *unbounded* iteration in general.

From Section 2.2, every Dice function of type $f : \tau \to \tau$ denotes a substochastic matrix and in turn, defines a DTMC. Conversely, every DTMC over a finite state-space can be encoded as a Dice program of type $\tau \to \tau$. The semantics of the Dice program f then defines a Markov chain on the state space $\llbracket \tau \rrbracket$. For a Markov chain with underlying transition matrix P, a vector π is a *stationary distribution vector* if $P\pi = \pi$. In many applications, Markov chains are useful when it can be shown that their behaviour closely approximates a given target within a reasonable number of steps in its time evolution. It is then useful to have a measure of how close a chain is from stationarity. For this purpose, the *mixing time* of a DTMC X with transition kernel P and stationary distribution π is often defined as $\min\{t : \max_{x \in \Omega} \|P^t(x, \cdot) - \pi\|_{TV} \leq 1/4\}$, where $\| \cdot \|_{TV}$ is the total variation distance. Intuitively, the mixing time gives the minimum amount of steps X must be iterated in order to be reasonably close to its limiting behaviour.

It may be tempting to sidestep the lack of an unbounded iteration operator by using $\texttt{iterate}(f, i, k)$ for some sufficiently large k in order to approximate the stationary distribution π of some program f. However, determining the mixing time of an arbitrary expression f remains a challenging task. While classic Markov chain theory asserts that iterates of a chain with distribution P approach its stationary distribution subject to mild conditions on its internal structure (such as non-periodicity), it does not provide bounds on the number of steps required to be sufficiently close. This makes the naïve approach to iteration – using a bounded loop – unsatisfactory, as it does not allow the user to ascertain whether the posterior is close enough to stationarity. Or rather, such an approach offsets the burden of knowing the mixing time to the user, rather than the compiler.

Moreover, if empirical exploration of the distribution obtained by `iterate`(f, i, k) for different values of k can give an indication of progress towards stationarity, there are DTMCs where the convergence rate in total-variation undergoes an abrupt change over a negligible period of time [11]. This phenomenon, called *cutoff*, makes it infeasible to rely on heuristics based on statistical distances between successive iterations to determine when to stop the computation in the hope of getting a distribution that is sufficiently close to stationarity.

▶ **Example 5** (Limitations of finite iteration)**.** To illustrate the challenges in determining the stationary distribution of a DTMC, we examine the Bernoulli-Laplace model of diffusion. This model can be depicted using two urns containing n balls each. Initially, the left and right urns contain n red balls and n blue balls, respectively. In each step, a ball is chosen at random from each urn, and they are exchanged. After sufficiently many exchanges, urns will be mixed, each of them containing approximately equal proportions of red and blue balls.

This process is fully determined by the number of blue (red) balls in one of the urns. Call this set $\Omega = \{1, \ldots, n\}$. The transition matrix of this model can be built by considering how many blue balls will be in the left urn given that there are $b \leq n$ currently. There are three possibilities: either a blue ball or a red ball is grabbed from both urns ($P_{b,b} = 2\frac{b(n-b)}{n^2}$), a red ball is grabbed from the left urn and a blue ball is grabbed from the right urn ($P_{b,b+1} = \left(\frac{n-b}{n}\right)^2$), a blue ball is grabbed from the left urn and a red ball is grabbed from the right urn ($P_{b,b-1} = \left(\frac{b}{n}\right)^2$). This process can be encoded in the following Dice program:

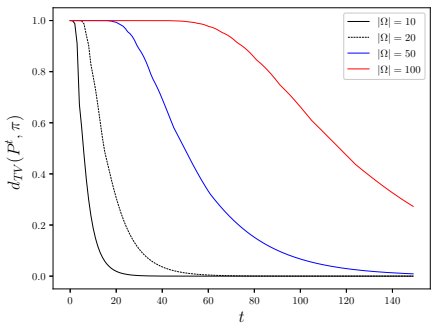
```
fun bernoulli_laplace(blues_in_urn : int) {
  let urn =
  if blues_in_urn == 0 then (0, 1, 0, 0, ...)
  else if blues_in_urn == 1 then
     (1/n^2, (2*(n-1))/n^2, (n-1)^2/n^2, 0, ...)
  else if blues_in_urn == 2 then
     (0, 2^2/n^2, (2*(n-2)*2)/n^2, (n-2)^2, ...)
  ...
  else (..., 0, 0, 1, 0) in urn
}
let ith_trial = iterate(bernoulli_laplace, 0, i) in ith_trial
```

Note that this idealized encoding of the Bernoulli-Laplace model assumes a fixed urn size n, where "..." indicates a distribution that ought to follow above's model. Concrete instances of this program can be found in the tests provided in our implementation.

Consider the problem of determining the convergence of the above program to its stationary state. To assess the number of iterations required for the distribution to reach a sufficiently close approximation to the stationary state, we can conduct multiple experiments using various urn sizes. Analysis of the model [13] yields that for any finite state space Ω, for any $b \in \Omega$, the stationary distribution is given by

$$\pi(b) = \binom{n}{b}\binom{n}{n-b}\Big/\binom{2n}{n}, \quad 0 \leq b \leq n.$$

We can measure the time it takes to become reasonably close to this distribution by calculating the total-variation between the distribution of the urn at each iteration step and its stationary state for different urn sizes. Note how the distance between the distribution and its stationary state remains close to its maximum value before dropping to near zero during a short period of time. This cutoff phenomenon shows the impracticality of approximate heuristics to determine the optimal number of iterations required for a probabilistic function to converge to a distribution that accurately represents its limiting behaviour.



Indeed, it has been proven that the Bernoulli-Laplace diffusion model exhibits a cutoff at time $\frac{1}{4} n \log n$ [13]. While a cutoff is not present in every DTMC, there are various families where it has been proven to exist [2, 11]. Furthermore, it remains a challenging task to determine if a chain has a cutoff window. For these reasons, a form of unbounded iteration is genuinely needed.

4 Introducing Unbounded Iteration

At first glance, it seems like unbounded iteration should force us out of the discrete probability setting by introducing nonterminating behaviour. How can potentially infinite execution runs be given a semantics with finitely-supported probability distributions? Perhaps surprisingly, we will now see that unbounded iteration can be accommodated in this setting.

If we think of a probabilistic program $t : \tau \to \tau$ as encoding the dynamics of some Markov chain, iterating t should give us the limiting behaviour of the chain, $\lim_{n\to\infty} [\![t]\!]^n$. However, this limit does not always exist, as is the case for example for chains that exhibit periodic behaviour. Thus, we cannot use it to define the semantics of unbounded iteration in general. Nonetheless, we can guarantee the existence of a limit by considering the *average time* spent on each of the possible states defined by the probabilistic function to be iterated: indeed, while the powers of an arbitrary stochastic matrix P need not have a limit, its *Cesàro sum* $\lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} P^k$ always exists [34, Section 8.4]. This is what we propose as semantics here. Note that this is not the first time that this is suggested as a reasonable semantics of iteration: in [8], the authors mention it in passing in the context of continuous probability. Here, we show that it gives a suitable semantics to iteration even in the discrete case. A precise way to compute it, together with a benchmark of our implementation against approximations that use bounded iteration are provided in Section 6.

Given a program $t : \tau \to \tau$, we want the program $\texttt{iterate}\ t$ to denote a distribution that represents the average time spent on each element of $[\![\tau]\!]$ by all finite iterations of t. The typing rule for this iterator is given by

$$\frac{\Gamma \vdash t : \tau \to \tau}{\Gamma \vdash \texttt{iterate}\ t\ : \tau}$$

Operationally, we can think of $\texttt{iterate}\ t$ as unfolding to the program $\mathsf{if}\ \mathsf{flip}\ \frac{1}{n}\ \mathsf{then}\ (\texttt{iterate}\ t)\ \mathsf{else}\ t^n$, assuming n is a runtime variable keeping track of the current execution step. Denotationally, the semantics of $\texttt{iterate}\ t$, is given by the Cesàro sum of $t : \tau \to \tau$, *i.e.*,

$$[\![\texttt{iterate}\ t]\!] := \left(\lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} [\![t]\!]^k \right)_0 , \text{ where } A_0 \text{ denotes the first column of the matrix } A.$$

The choice of 0 here is arbitrary since, as we will see, all the columns of the Cesàro sum are equal. If we think of t as encoding the transition kernel of some Markov chain, this limit characterises the fraction of time the underlying Markov chain spends in each of its possible states, and its value is independent of the initial distribution we consider. The intuition is that the averaging of the Cesàro sum removes any oscillatory behaviour that could prevent the sequence of iterated powers to converge. The following example illustrates this point.

▶ **Example 6.** Consider the following program, one of the simplest programs exhibiting periodic behaviour (note that the exclamation mark denotes negation).

```
fun swap_state(a : bool) : bool { !a }
iterate swap_state
```

This program denotes a Markov chain that jumps between its only two states at every step. It is not hard to see that this chain has stationary distribution $\begin{pmatrix}1/2 & 1/2\end{pmatrix}^T$. Intuitively, if we distribute the total weight equally between the two states, then swapping them does not change the distribution. It should be noted that the periodic behaviour of the program makes it impossible to arrive at such distribution through successive iterations. Indeed, iterating the chain will simply shift the weight back and forth between the two states, while its Cesàro sum is equal to the stationary distribution.

4.1 Exact Inference

It is clear that to obtain $[\![\texttt{iterate}\ t]\!]$, we could simply compute $\frac{1}{n}\sum_{k=1}^{n}[\![t]\!]^k$ for successive values of n. However, the compiler would then need to decide when the approximation is sufficiently close to the limit to stop the computation. There are several possible heuristics to decide when a given approximation is satisfactory. For example, one could decide to stop when the distance between a given approximation and the previous one differ only by a small value ε in KL divergence. As we aim for exact inference, this approach would be insufficient, not to mention that it suffers from the same drawbacks as bounded iteration, highlighted in Section 3. It is in general a hard problem to determine whether the underlying Markov chain exhibits cutoff and when, thereby making it difficult to decide when we are reasonably close to the asymptotic behaviour of the chain. We can however sidestep these challenges by leveraging standard results from linear algebra that suggest a different–and most importantly, *exact*–way to compute the semantics of iteration, which we now explain.

Given any scalar sequence $(a_n)_{n\in\mathbb{N}}$, there is an associated *Cesàro* sequence of its averages, $(b_n)_{n\in\mathbb{N}}$, where $b_n = \frac{1}{n}\sum_{k=0}^{n-1} a_k$; we say that the sequence $(a_n)_{n\in\mathbb{N}}$ is *summable* to s (or just summable) whenever $\lim_{k\to\infty} b_k = s$. The same idea can easily be transported to the setting of matrices: given a sequence $(A^n)_{n\in\mathbb{N}}$ of powers of some square matrix A, we say that A is *convergent* when $\lim_{k\to\infty} A^k$ exists and that A is *summable* whenever its sequence of averages $\left(\frac{1}{n}\sum_{k=0}^{n-1} A^k\right)_{n\in\mathbb{N}}$ converges. Note that matrices that do not converge can be summable. Indeed, the averaging of a Cesàro sequence smooths away any periodic behaviour the chain could have. Furthermore, whenever a matrix A converges, its limit coincides with that of its Cesàro sequence [34]. In order for the Cesàro sum to be a suitable candidate for the semantics of `iterate` t, we must first make sure that this limit always exists for the family of matrices we are interested in. It is a well-known fact that this is always the case for (sub)stochastic matrices.

The following result about *substochastic* matrices is a mild generalisation of a theorem appearing in [34, Section 8.4] for stochastic ones. We will use it in Section 6, where we explain how our implementation computes the relevant projection and evaluate its performance.

▶ **Proposition 7.** *Every substochastic matrix M is summable and its sum is equal to the projection onto* $\mathsf{Ker}(\mathbf{I} - M)$ *along* $\mathsf{Im}(\mathbf{I} - M)$.

Interestingly, when the limit $\lim_n M^n$ exists, it is also equal to the projection onto $\mathsf{Ker}(\mathbf{I} - M)$ along $\mathsf{Im}(\mathbf{I} - M)$ and is therefore equal to the Cesàro sum [34, Section 7.10]. While we do not use this fact, it lends further support to the intuition that the Cesàro sum provides a generalised semantics for iteration, even in the context when the limit of finite iterations does not exist.

We make use of Proposition 7 when explaining how our implementation computes the relevant projection and evaluate its performance in Section 6.

5 An Alternative: Least Fixpoint Iterator

We can give an alternative form of iteration which is more closely related to imperative while-loops in the style of [29]. This follows the approach taken in [43], where iteration is defined by exploiting coproducts: starting from a program $t : A \to A + B$, $\mathsf{iterate}\ t\ \mathsf{from}\ u$ repeatedly calls t (starting from some value u) until the output of t is of type B, signalling the stopping condition for iteration, and returning the last output of t.

Although Dice does not have explicit sum types, the same procedure can be mimicked using products and the fact that $\mathbb{B} = 1 + 1$. We can represent the repeated application of some expression $t : \tau \to \mathbb{B} \times \tau$ that depends on an argument $x : \tau$ through an iterator that repeatedly calls t starting from some initial value u, in each step consuming a Boolean variable (that acts as a guard) until this variable evaluates to false, in which case the value of the last evaluation is output. The derived typing rule for this iterator is given by

$$\frac{\Gamma \vdash u : \tau \qquad \Gamma \vdash t : \tau \to \mathbb{B} \times \tau}{\Gamma \vdash \mathtt{iterate}_0\ t\ \mathsf{from}\ u : \tau}$$

The operational intuition given above suggests that $\mathtt{iterate}_0\ t\ \mathsf{from}\ u$ unfolds once to $\mathsf{let}\ (b, v) = t(u)\ \mathsf{in}\ (\mathsf{if}\ b\ \mathsf{then}\ (\mathtt{iterate}_0\ t\ \mathsf{from}\ v)\ \mathsf{else}\ v)$. In plain language, we apply t once to u, getting a pair (b, v) of a Boolean guard and a new value v; we then examine b to decide whether to apply t again or to return v.

We can think of this process in two ways: as iterating t from u or, equivalently, as iterating t and applying the resulting operation to u. With the latter perspective, we want to find a (sub)stochastic matrix M such that $M[\![u]\!] = [\![\mathtt{iterate}_0\ t\ \mathsf{from}\ u]\!]$. Then,

$$\begin{aligned} M[\![u]\!] &= [\![\mathtt{iterate}_0\ t\ \mathsf{from}\ u]\!] \\ &= [\![\mathtt{let}\ (b, v) = t(u)\ \mathtt{in}\ (\mathtt{if}\ b\ \mathtt{then}\ (\mathtt{iterate}_0\ t\ \mathsf{from}\ v)\ \mathtt{else}\ v)]\!] \\ &= \mathsf{join}_\tau \circ (\mathsf{id}_{\mathbb{B}} \otimes M \otimes \mathsf{id}_\tau) \circ (\mathsf{id}_{\mathbb{B}} \otimes \mathsf{copy}_\tau) \circ [\![t]\!][\![u]\!] \end{aligned}$$

Since u can denote an arbitrary distribution, we are looking for a matrix M that is a solution of the equation above. This equation might have several solutions. However, in the presence of an order, it is traditional in programming language semantics to take the smallest solution as the canonical semantics of iteration [46]. For this purpose, we equip the set of substochastic maps with the pointwise order, *i.e.*, $A \leq B$ for two substochastic matrices if $Ax \leq Bx$ for all nonnegative $x \in \mathbb{R}^n$. With this order, we want the semantics of $\mathsf{iterate}_0\ t$ to be the least fixpoint of the map φ given by $\varphi(M) = \mathsf{join}_\tau \circ (\mathsf{id}_{\mathbb{B}} \otimes M \otimes \mathsf{id}_\tau) \circ (\mathsf{id}_{\mathbb{B}} \otimes \mathsf{copy}_\tau) \circ [\![t]\!]$ for $M : [\![\tau]\!] \to [\![\tau]\!]$. Note that $\mathsf{id}_{\mathbb{B}} = [\![\mathsf{true}]\!][\![\mathsf{true}]\!]^T + [\![\mathsf{false}]\!][\![\mathsf{false}]\!]^T$, so that

$$\begin{aligned} \varphi(M) &= \mathsf{join}_\tau \circ \left(\left([\![\mathsf{true}]\!][\![\mathsf{true}]\!]^T + [\![\mathsf{false}]\!][\![\mathsf{false}]\!]^T\right) \otimes M \otimes \mathsf{id}_\tau\right) \circ (\mathsf{id}_{\mathbb{B}} \otimes \mathsf{copy}_\tau) \circ [\![t]\!] \\ &= \mathsf{join}_\tau \circ \left(\left([\![\mathsf{true}]\!][\![\mathsf{true}]\!]^T\right) \otimes M \otimes \mathsf{id}_\tau\right) \circ (\mathsf{id}_{\mathbb{B}} \otimes \mathsf{copy}_\tau) \circ [\![t]\!] \\ &\quad + \mathsf{join}_\tau \circ \left(\left([\![\mathsf{false}]\!][\![\mathsf{false}]\!]^T\right) \otimes M \otimes \mathsf{id}_\tau\right) \circ (\mathsf{id}_{\mathbb{B}} \otimes \mathsf{copy}_\tau) \circ [\![t]\!] \end{aligned}$$

Since $\mathsf{join}_X(\delta_{\mathsf{true}}, x, y) = x$ and $\mathsf{join}_X(\delta_{\mathsf{false}}, x, y) = y$, we can simplify the previous expression considerably: $\varphi(f) = M \circ (\llbracket \mathsf{true} \rrbracket^T \otimes \mathsf{id}_\tau) \circ \llbracket t \rrbracket + (\llbracket \mathsf{false} \rrbracket^T \otimes \mathsf{id}_\tau) \circ \llbracket t \rrbracket$. Thus, φ is an affine transformation over the vector space of linear maps $\mathbb{R}^{\llbracket \tau \rrbracket} \to \mathbb{R}^{\llbracket \tau \rrbracket}$ (which is isomorphic to $\mathbb{R}^{\llbracket \tau \rrbracket} \otimes \mathbb{R}^{\llbracket \tau \rrbracket}$), given by $\varphi(M) = MB + A$, where $A := (\llbracket \mathsf{false} \rrbracket^T \otimes \mathsf{id}_\tau) \circ \llbracket t \rrbracket$ and $B := (\llbracket \mathsf{true} \rrbracket^T \otimes \mathsf{id}_\tau) \circ \llbracket t \rrbracket$. Moreover, φ maps substochastic maps to substochastic maps, since $\varphi(M)$ can also be expressed as $\mathsf{join}_\tau \circ (\mathsf{id}_{\mathbb{B}} \otimes M \otimes \mathsf{id}_\tau) \circ (\mathsf{id}_{\mathbb{B}} \otimes \mathsf{copy}_\tau) \circ \llbracket t \rrbracket$, which is a composition of substochastic maps, and therefore substochastic.

In the literature [8, 29], the problem of characterising this fixpoint is usually approached by showing that φ is order-continuous – then, its least fixpoint can be obtained by taking the suprema of successive iteration of φ starting from the bottom of the order, *i.e.*, the uniform zero distribution:

$$\bigvee_{n \geq 0} \varphi^n(0) = \lim_{n \to \infty} \sum_{k=0}^{n} A \circ B^k$$

However, in general, this only gives a recipe to *approximate* the desired least fixpoint. In the discrete setting that is ours, we want to compute it *exactly*. We lay out the details on how we accomplish this to Section 6, where we elucidate the internal implementation of this iterator and evaluate its performance.

6 Implementation and Empirical Evaluation

We now extend the core syntax of Dice by incorporating the two iterators described in Sections 4 and 5 [45]. We evaluate their performance by comparing the average time required to attain the stationary distribution of a specific probabilistic function using these, contrasted with the time needed to approximate the same distribution through bounded iteration.

Dice is implemented in OCaml and uses rsdd [22], a Rust-based implementation for building and handling decision diagrams, as its backend for compiling binary decision diagrams (BDDs). To extend Dice, we employ py.ml [32], an OCaml library that facilitates the dynamic binding of Python modules, to harness efficient and extensively used linear algebra tools for the computation of linear operators corresponding to the semantics given by the iterators. While at compile time a Dice program is encoded as a weighted BDD, we obtain its matrix representation by evaluating its action on the canonical Dirac basis. This matrix is subsequently sent through py.ml to a Python library, where the distribution associated to either $\mathsf{iterate}$ or $\mathsf{iterate}_0$ is obtained. Subsequently, we obtain the weighted BDD representation of this distribution by leveraging Dice's internal encoding [24, Section 5.1] to define a distribution over a finite set of states. All our experiments were ran single-threaded on the same machine with a 2.7GHz CPU and 15GB of RAM. Timings were recorded using time, the standard GNU utility.

6.1 Cesàro Iterator

For computing the semantics of `iterate` t we work with the matrix representation of $\llbracket t \rrbracket$. From the discussion in Section 4 and Proposition 7, we know that the semantics of `iterate` t is any column of the projection onto $\mathsf{Ker}(\mathbf{I} - \llbracket t \rrbracket)$ along $\mathsf{Im}(\mathbf{I} - \llbracket t \rrbracket)$.

There are several ways to compute this projection. We have chosen to use a *full-rank factorisation* of $\mathbf{I} - \llbracket t \rrbracket$. A full-rank factorisation of a $n \times n$ matrix M is a pair of of an $n \times r$ matrix B and a $r \times n$ matrix C such that $M = BC$, where r is the rank of B, C, and M. Given such a factorisation, the projection onto $\mathsf{Im}\, M$ along $\mathsf{Ker}\, M$ is given by $P = B(CB)^{-1}C$, a result which can be found in § 7.10 of [34]. Since $\mathbf{I} - P$ is the complementary projector of P, the projection onto $\mathsf{Ker}\, M$ along $\mathsf{Im}\, M$ is therefore $\mathbf{I} - B(CB)^{-1}C$.

In practice, we obtain a full rank factorisation from the singular value decomposition (SVD) of M, by taking $M = U \begin{pmatrix} \Sigma & 0 \\ 0 & 0 \end{pmatrix} V^T = (U_0|U_1) \begin{pmatrix} \Sigma & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} V_0 \\ V_1 \end{pmatrix}$, making $(U_0\Sigma)V_0 = U_0(\Sigma V_0)$ a full-rank factorisation of M, where U_0 corresponds to the matrix made by the first n columns of U where n is the number of non-zero singular values of Σ and, similarly, V_0 corresponds to the first n rows of V. Thus, $[\![\texttt{iterate}\ t]\!]$ is (say) the first column of $\mathbf{I} - U_0(V_0U_0)^{-1}V_0$, where U_0, V_0 come from the full-rank factorisation of $\mathbf{I} - [\![t]\!]$ given by its SVD (note that the singular values are not needed, since they cancel out for these choices of $\mathbf{B} = U_0\Sigma$ and $C = V_0$).

▶ **Example 8.** Consider the transition kernel $[\![t]\!]$ of the program that simply swaps two states (Example 6). An SVD of $M = \mathbb{I} - [\![t]\!]$ gives $M = U\Sigma V^T$ where $U = V = \begin{pmatrix} -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}$ and $\Sigma = \begin{pmatrix} 2 & 0 \\ 0 & 0 \end{pmatrix}$. Thus, $[\![\texttt{iterate}\ t]\!]$ is any column of $\mathbf{I} - U_0(V_0U_0)^{-1}V_0$, which gives the distribution $\begin{pmatrix} \frac{1}{2} & \frac{1}{2} \end{pmatrix}^\top$. This witnesses the fact that the program oscillates between its only two states, spending equal time on each. Note that this gives a stationary distribution of the corresponding Markov chain which cannot be obtained by successive iterations.

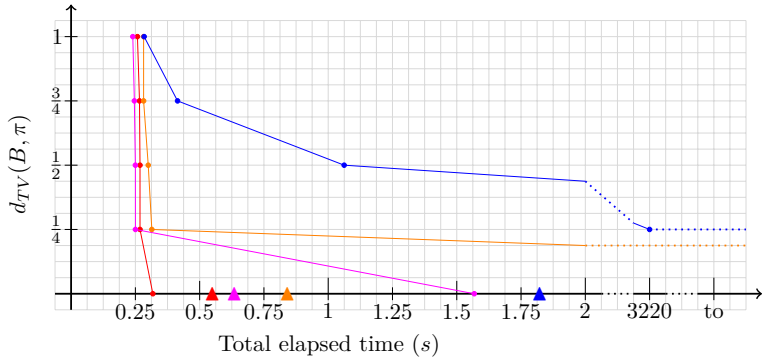


Figure 2 Comparative graph depicting the required time for various simulations of the Bernoulli-Laplace model to converge to a posterior distribution within $0 + \epsilon$, $\frac{1}{4}$, $\frac{1}{2}$, $\frac{3}{4}$ and 1 in total variation distance (drawn as points "•" in the above graph) from the stationary distribution, and the time it takes for $[\![\mathsf{iterate}\ t]\!]$ to compute the stationary distribution for an instance of the Bernoulli-Laplace model with equal number of states (drawn as triangles "▲"). Different colors represent distinct instances of the Bernoulli-Laplace model with different urn size. We extend the time line with a timeout data point (to) for the cases in which no posterior could be retrieved.

Empirical Evaluation. We now evaluate the time performance of iterate. Note that a direct comparison of our iterative operator with other existing forms of iteration in Dice is not feasible. This is because while our iterator obtains the exact posterior distribution of some function $t : \tau \to \tau$, Dice can only approximate this distribution by simulating the underlying transition kernel of t by unfolding its BDD representation a fixed number of times.

To empirically evaluate the performace of $[\![\mathsf{iterate}]\!]$, we compare the time it takes for $[\![\mathsf{iterate}\ t]\!]$ to obtain the stationary distribution π for t an instance of the Bernoulli-Laplace diffusion model described in Section 3, with urns of various sizes $|\Omega| = \{3, 7, 10, 20\}$, and the time it takes for an iterative simulation to obtain a posterior distribution B at distances

of $1, \frac{3}{4}, \frac{1}{2}, \frac{1}{4}$ and $0 + \varepsilon$ in total variation from the stationary distribution π. Already for urns as small as $|\Omega| = 7$ it is not possible to obtain a posterior that is within $\varepsilon = 0.001$ distance from π in any reasonable amount time. Moreover, for models with an urn size of 10 or more, it becomes practically infeasible to achieve a posterior distribution beyond the mixing time of t. For this reason, we have decided to set $\epsilon = 0.005$ for the above experiments.

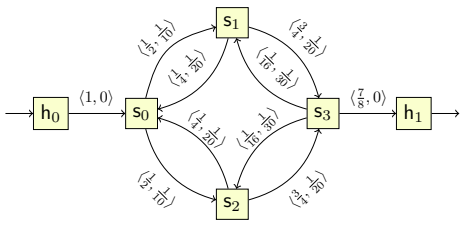
Colored dots in the graph in Fig. 2 represent the time it takes for the bounded iterative method to initially produce a posterior B that is within y-axis distance from π, while the colored triangles placed along the x-axis indicate the time required for $[\![\mathsf{iterate}\ t]\!]$ to compute the exact posterior, π. It can be observed that even in experiments with small urn sizes, achieving a reasonably close approximation to the stationary distribution (reaching its mixing time) is not feasible through existing means.

6.2 Least Fixpoint Iterator

As introduced in Section 5, the semantics of $\mathtt{iterate}_0\ t\ \mathsf{from}\ u$ is given by the fixpoint of equation (**??**). One way to compute this fixpoint *exactly* is with linear programming. Every fixpoint M of φ will satisfy $MA + B = M$ so we are looking for $M \in \mathbb{R}^{[\![\tau]\!]} \otimes \mathbb{R}^{[\![\tau]\!]}$ that satisfies $M(\mathbf{I} - A) = B$ and $M \geq 0$. We also need a linear form to minimise such that the solution to the corresponding linear program is the desired least fixpoint. In other words, we are looking for $C \in \mathbb{R}^{[\![\tau]\!]} \otimes \mathbb{R}^{[\![\tau]\!]}$ such that the minimum of $C^T M$ subject to the linear constraints $M(\mathbf{I} - A) = B$ and $M \geq 0$ is the least fixpoint of φ. Which C should we choose? We claim that any strictly positive C will work.

▶ **Proposition 9.** *Let C be any positive vector of $\mathbb{R}^{[\![\tau]\!]} \otimes \mathbb{R}^{[\![\tau]\!]}$. The least fixpoint of $\varphi : M \mapsto MA + B$ over the poset of substochastic maps is the solution to the following linear program: minimise $C^T M$ subject to the constraints $M(\mathbf{I} - A) = B$ and $M \geq 0$.*

Empirical Evaluation. Now we compare the efficiency of $\mathsf{iterate}_0\ t\ \mathsf{from}\ u$ to obtain the least fixpoint of a probabilistic procedure against the time it takes to approximate the same distribution through finite iterative means. To this end, we adapt a probabilistic program that models a packet delivery reliability problem, inspired by Bayonet [15], a domain-specific language (DSL) focusing in network analysis, which captures the inherent probabilistic behaviour of computer networks. Similar to Dice, Bayonet allows to define discrete distributions within its program structure. This is achieved through PSI in its backend [16], a discrete PPL featuring a symbolic engine that generates concise expressions for representing distributions. A network topology defines the network nodes which can be hosts or switches, and the links that interconnect them. Every link between two nodes is represented by two ports, and each of these nodes has both an input and an output queue, for received and outgoing packets, respectively. To capture the behaviour of a node, we specify how the node processes the packets it receives.



Consider above's network topology, where some switches s_i have a non-zero probability $\langle -, q\rangle$ of dropping a packet at any given moment and forwarding their packet with some probability $\langle p, -\rangle$. Host h_0 sends a packet to host h_1 every clock tick. Switch s_0 forwards

half of h_0's packets to s_1 and the other half to s_2, with a $\frac{1}{10}$ chance of dropping a packet. Both s_1 and s_2 forward their packets to s_3 with a $\frac{3}{4}$ chance and send it back with a $\frac{1}{4}$ chance. At every moment, they have a $\frac{1}{20}$ chance of dropping the packet. Finally, s_3 has a $\frac{7}{8}$ chance of forwarding its packet to h_1 without any inconvenience and a $\frac{1}{8}$ chance of sending it back to either s_1 or s_2 with equal probability and a $\frac{1}{30}$ chance of dropping it.

The problem of packet delivery reliability is defined as the probability that a given packet (or flow of packets) reaches its intended destination. In our setting, we are interested in knowing the reliability of the entire network. This is, for any packet sent by h_0, what is the probability that it hits h_1. This network's behaviour is captured by the following program:

```
fun network(s_i) {
  if s_0 then (if flip 1/2 then (s_1, flip 1/10) else (s_2, flip 1/10)) else
  if s_1 then (if flip 3/4 then (s_3, flip 1/20) else (s_0, flip 1/20)) else
  if s_2 then (if flip 3/4 then (s_3, flip 1/20) else (s_0, flip 1/20)) else
  if flip 7/8 then (s_3, false) else
    (if flip 1/2 then (s_1, flip 1/30) else (s_2, flip 1/30))
}
fun aux(a) { let x = network(a) in (if (π_2 t) then (aux (π_1 t)) else (π_1 t)) }
```

The `network` function returns, for any state s, the distribution over the state space that represents the probability of transitioning to any other state. To evaluate the efficiency of $[\![\mathsf{iterate}_0\ t\ \mathsf{from}\ u]\!]$ in computing the fixpoint of the transition function implicitly given by the above program, we compare the time it takes to obtain this distribution via the least fixpoint iterator against the time it takes to *approximate it* through successive iterations, by unrolling the underlying BDD. Since, similar to the previous example, we cannot directly compare these two times, as one method finds the desired distribution exactly and the other only approximates it, we also provide how close we are in total variation distance to the target distribution π at each iteration i.

We denote by f^i the distribution given by the first projection of the output provided by $\texttt{iterate}(\texttt{aux}, s_0, i)$. Each column of the following table displays the average time taken by five different executions of f^i to obtain the posterior, along with its proximity to π, the stationary distribution of the transition given by `network`:

# iterations	1	3	5	7	9
$d_{TV}(f^i, \pi)$	0.912627	0.168776	0.023505	0.008805	0.013332
Ellapsed time (s)	0.252	0.266	0.296	1.030	58.32

While the simulation quickly converges to stationary, the oscillatory behaviour of the model hinders the possibility of establishing any stopping criterion. This naïve iterative method becomes unwieldy, already taking one minute to simulate only nine steps of the network. We contrast the times in the above table with the 0.54 seconds that it takes $\texttt{iterate}_0$ `network` from s_0 to compute the stationary distribution.

7 Conclusion and Future Work

We have shown that it is possible to extend discrete probabilistic programming with unbounded iteration in at least two different ways, without giving up exact inference. Unbounded iteration is valuable because it allows us to compute the limiting behaviour of probabilistic programs that encode discrete-time stochastic processes. Since deciding when a stochastic process is sufficiently close to its limiting distribution is intractable in principle, we have argued that unbounded iteration and exact inference are both necessary in this setting. Finally, we have implemented these two different forms of iteration and demonstrated their performance on simple benchmarks, extending a functional programming language with discrete probabilistic choices.

Comparing the two iterators. In this paper, we have given two forms of unbounded iteration: one based on averaging the behaviour of a term $t : \tau \to \tau$ over all iterations, and another given by iterating a term $t : \tau \to \mathbb{B} \times \tau$ from a given initial condition u, while the Boolean guard of t remains true. The former only makes sense in the context of probabilistic programming while the latter is a direct translation of while loops into the setting of a (functional) PPL.

While we have not given the Cesàro iterator as a fixpoint, it can also be characterised as one. Indeed, it is a well-known corollary of Proposition 7 that the Cesàro sum of a (sub)stochastic map M always gives a stationary distribution for the Markov chain it represents, in other words a fixpoint of M. Indeed, if P is the projection onto $\mathsf{Ker}(\mathbf{I} - M)$ along $\mathsf{Im}(\mathbf{I} - M)$, then for all x, $Px - MPx = 0$ and therefore $MPx = Px$.

At first sight, comparing these two forms of iteration is not meaningful, since they take arguments of different types. Therefore, one cannot simply ask when is $\texttt{iterate}\ t = \texttt{iterate}_0\ t\ \mathsf{from}\ u$, since the same t cannot be fed to both. Nevertheless, given $t : \tau \to \tau$, one can ask when does $\texttt{iterate}\ t = \texttt{iterate}_0\ (t, \texttt{true})\ \mathsf{from}\ u$, for some program u? It turns out that if the guard is always $\texttt{true}$, the semantics of $\texttt{iterate}_0\ (t, \texttt{true})\ \mathsf{from}\ u$ is uniformly 0 (regardless of u), which is not necessarily the case for $\texttt{iterate}\ t$. Conversely, given $t : \tau \to \mathbb{B} \times \tau$, one can obtain a program $t' : \tau \to \tau$ by marginalising over the Boolean guard. Even in this case, $\texttt{iterate}\ t'$ does not necessarily coincide with $\texttt{iterate}_0\ t\ \mathsf{from}\ u$. There are edge cases for which the two are equal (*e.g.*, when the spectral radius of t' is strictly less than 1, see Appendix A.2), but they do not correspond to any meaningful conditions on the corresponding program. Therefore, the forms of iteration have genuinely distinct capabilities and use cases: the first is useful to obtain the stationary distribution of some program encoding the dynamics of a Markov chain, while the second can be used to simulate while loops.

Related work. As far as we are aware, the only other line of work that combines exact inference and unbounded iteration is [6]. There, the semantics of programs is given in terms of probability generating functions and thus rather different from ours, making a direct comparison with our work more challenging. Moreover, while their language allows some infinitely-supported distributions, it does not allow conditioning inside of loops. In [7], the authors present an imperative PPL for discrete random variables, called BernoulliProb. Notably, BernoulliProb has while-loops and therefore support unbounded iteration. While the semantics of while-loops is closely related to the one we give to the least fixpoint iterator of Section 5, in practice, the implementation of [7] only approximates this least fixpoint using KL-divergence as a measure of convergence. Thus, their implementation suffers from the limitations we have highlighted in Section 3.

In [16], the authors introduce PSI, a PPL that is able to perform exact symbolic inference for programs with discrete and continuous random variables. However, the syntax of their language allows only bounded iteration. There are other general-purpose PPLs and frameworks including looping constructs with constant bounds [19, 36], leading to approximate rather than exact inference. These languages do not offer guarantees on the convergence rates of the associated fixpoint computation, running into the problems outlined in Section 3.

It is also worth comparing our work with the the Probabilistic Model Checking (PMC) tools [30, 10], which focus on generalising traditional model checking to verify that a probabilistic system satisfies a particular temporal logic formula. The main difference between Dice and these tools lies in their approach to probabilistic inference. While Dice utilizes WMC over BDDs, PMC tools frequently employ ADDs and explicit matrix representations that are combined with iterative techniques to derive posterior distributions. For discrete-time

Markov chains, these tools integrate PCTL operators whose semantics is akin to the least fixed-point operator of this paper [20], allowing for the expression of stationary distributions, when they exist. Although obtaining the exact stationary distribution is feasible, practical implementations often resort to iterative methods to address issues stemming from state space explosion – similar to those faced by Dice – making them prone to the challenges discussed in Section 3.

Finally, there are domain-specific PPLs that include some form of unbounded iteration [41, 42], but do so only for their restricted domain of application. In particular, the more narrow focus of [42] on network specification gives stronger guarantees on the convergence of the loops that can be specified in the language. This allows for exact inference in the presence of unbounded iteration, albeit in a setting which is less general than ours.

Future work. As mentioned in Remark 3, our work could be formulated in a more categorical context. This would enable us to connect the different notions of iteration and their corresponding fixpoints to traces in monoidal categories. In categories for which the monoidal product coincides with the categorical product, a trace always defines a notion of fixpoint. This is not necessarily the case in more general monoidal categories, such as that of (sub)stochastic maps. Finding a suitable trace would open the possibility of reasoning equationally about probabilistic programs with iteration.

In a more practical note, one possible avenue for future work is to explore data structures that enable the representation of discrete probabilistic programs with loops without incurring in any kind of space explosion. Similarly, efficient transformations from explicit representations of DTMCs to BDDs have been explored in the context of finite-horizon Markov Chains [23], a natural next step is to adapt the insights gained for finite iterations to a compact representation of iterators.

References

1 Gilles Barthe, Cédric Fournet, Benjamin Grégoire, Pierre-Yves Strub, Nikhil Swamy, and Santiago Zanella-Béguelin. Probabilistic relational verification for cryptographic implementations. *SIGPLAN Not.*, 49(1):193–205, January 2014. doi:10.1145/2578855.2535847.

2 Riddhipratim Basu, Jonathan Hermon, and Yuval Peres. Characterization of cutoff for reversible markov chains. *The Annals of Probability*, 45(3), May 2017. doi:10.1214/16-aop1090.

3 Randal E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Comput. Surv.*, 24(3):293–318, September 1992. doi:10.1145/136035.136043.

4 Dmitry Bugaychenko. On application of multi-rooted binary decision diagrams to probabilistic model checking. In *Proceedings of the 13th International Conference on Verification, Model Checking, and Abstract Interpretation*, VMCAI'12, pages 104–118, Berlin, Heidelberg, 2012. Springer-Verlag. doi:10.1007/978-3-642-27940-9_8.

5 Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *Artificial Intelligence*, 172(6):772–799, 2008. doi:10.1016/j.artint.2007.11.002.

6 Mingshuai Chen, Joost-Pieter Katoen, Lutz Klinkenberg, and Tobias Winkler. Does a program yield the right distribution? verifying probabilistic programs via generating functions. In *International Conference on Computer Aided Verification*, pages 79–101. Springer, 2022.

7 Guillaume Claret, Sriram K. Rajamani, Aditya V. Nori, Andrew D. Gordon, and Johannes Borgström. Bayesian inference using data flow analysis. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, pages 92–102, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2491411.2491423.

8 Fredrik Dahlqvist, Alexandra Silva, and Dexter Kozen. *Semantics of Probabilistic Programming: A Gentle Introduction*, pages 1–42. Cambridge University Press, 2020. doi:10.1017/9781108770750.002.

9 Swaraj Dash, Younesse Kaddar, Hugo Paquet, and Sam Staton. Affine monads and lazy structures for bayesian programming. *Proc. ACM Program. Lang.*, 7(POPL), January 2023. doi:10.1145/3571239.

10 Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A storm is coming: A modern probabilistic model checker, 2017. arXiv:1702.04311.

11 P Diaconis. The cutoff phenomenon in finite markov chains. *Proceedings of the National Academy of Sciences of the United States of America*, 93(4):1659–1664, February 1996. doi:10.1073/pnas.93.4.1659.

12 Persi Diaconis. The markov chain monte carlo revolution. *Bulletin of the American Mathematical Society*, 46(2):179–205, November 2008. doi:10.1090/s0273-0979-08-01238-x.

13 Persi Diaconis and Mehrdad Shahshahan. Time to reach stationarity in the bernoulli-laplace diffusion model. *Siam Journal on Mathematical Analysis*, 18:208–218, 1987.

14 Nate Foster, Dexter Kozen, Konstantinos Mamouras, Mark Reitblatt, and Alexandra Silva. Probabilistic netkat. In Peter Thiemann, editor, *Programming Languages and Systems*, pages 282–309, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

15 Timon Gehr, Sasa Misailovic, Petar Tsankov, Laurent Vanbever, Pascal Wiesmann, and Martin Vechev. Bayonet: Probabilistic inference for networks. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2018, pages 586–602, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3192366.3192400.

16 Timon Gehr, Sasa Misailovic, and Martin Vechev. Psi: Exact symbolic inference for probabilistic programs. In Swarat Chaudhuri and Azadeh Farzan, editors, *Computer Aided Verification*, pages 62–83, Cham, 2016. Springer International Publishing.

17 Noah D. Goodman, Vikash K. Mansinghka, Daniel Roy, Keith Bonawitz, and Joshua B. Tenenbaum. Church: A language for generative models. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence*, UAI'08, pages 220–229, Arlington, Virginia, USA, 2008. AUAI Press.

18 Andrew D. Gordon, Thomas A. Henzinger, Aditya V. Nori, and Sriram K. Rajamani. Probabilistic programming. In *Proceedings of the on Future of Software Engineering*, pages 167–181. ACM, 2014. doi:10.1145/2593882.2593900.

19 Maria I. Gorinova, Andrew D. Gordon, and Charles Sutton. Probabilistic programming with densities in SlicStan: efficient, flexible, and deterministic. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–30, January 2019. doi:10.1145/3290348.

20 Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Form. Asp. Comput.*, 6(5):512–535, September 1994. doi:10.1007/BF01211866.

21 Holger Hermanns, Marta Kwiatkowska, Gethin Norman, David Parker, and Markus Siegle. On the use of mtbdds for performability analysis and verification of stochastic systems. *The Journal of Logic and Algebraic Programming*, 56(1):23–67, 2003. Probabilistic Techniques for the Design and Analysis of Systems. doi:10.1016/S1567-8326(02)00066-8.

22 Steven Holtzen. RSDD. https://github.com/neuppl/rsdd/, 2023.

23 Steven Holtzen, Sebastian Junges, Marcell Vazquez-Chanlatte, Todd Millstein, Sanjit A. Seshia, and Guy Van Den Broeck. Model checking finite-horizon markov chains with probabilistic inference, 2021. arXiv:2105.12326.

24 Steven Holtzen, Guy Van den Broeck, and Todd Millstein. Scaling exact inference for discrete probabilistic programs. *Proc. ACM Program. Lang.*, 4(OOPSLA), November 2020. doi:10.1145/3428208.

25 Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, 1985. doi:10.1017/CBO9780511810817.

26 Joe Hurd. Formal verification of probabilistic algorithms. Technical Report UCAM-CL-TR-566, University of Cambridge, Computer Laboratory, May 2003. doi:10.48456/tr-566.

27 Joost-Pieter Katoen. Model checking meets probability: A gentle introduction. In Manfred Broy, Doron A. Peled, and Georg Kalus, editors, *Engineering Dependable Software Systems*, volume 34 of *NATO Science for Peace and Security Series, D: Information and Communication Security*, pages 177–205. IOS Press, 2013. doi:10.3233/978-1-61499-207-3-177.

28 D. Knuth and A. Yao. *Algorithms and Complexity: New Directions and Recent Results*, chapter The complexity of nonuniform random number generation. Academic Press, 1976.

29 Dexter Kozen. Semantics of probabilistic programs. *Journal of Computer and System Sciences*, 22(3):328–350, 1981. doi:10.1016/0022-0000(81)90036-2.

30 Marta Kwiatkowska, Gethin Norman, and David Parker. Prism 4.0: Verification of probabilistic real-time systems. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification*, pages 585–591, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

31 D.A. Levin and Y. Peres. *Markov Chains and Mixing Times*. MBK. American Mathematical Society, 2017.

32 Thierry Martinez. pyml. https://opam.ocaml.org/packages/pyml/, 2023.

33 A.J. Menezes, J. Katz, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*. Discrete Mathematics and Its Applications. CRC Press, 1996.

34 Carl D. Meyer. *Matrix Analysis and Applied Linear Algebra*. Society for Industrial and Applied Mathematics, USA, 2000.

35 T. Minka, J.M. Winn, J.P. Guiver, Y. Zaykov, D. Fabian, and J. Bronskill. /Infer.NET 0.3, 2018. Microsoft Research Cambridge. http://dotnet.github.io/infer.

36 T. Minka, J.M. Winn, J.P. Guiver, Y. Zaykov, D. Fabian, and J. Bronskill. /Infer.NET 0.3, 2018. Microsoft Research Cambridge. http://dotnet.github.io/infer.

37 Praveen Narayanan, Jacques Carette, Wren Romano, Chung-chieh Shan, and Robert Zinkov. Probabilistic inference by program transformation in Hakaru (system description). In Oleg Kiselyov and Andy King, editors, *Functional and Logic Programming*, pages 62–79, Cham, 2016. Springer International Publishing.

38 Frank Noé. Probability distributions of molecular observables computed from Markov models. *The Journal of Chemical Physics*, 128(24):244103, June 2008. doi:10.1063/1.2916718.

39 Fritz Obermeyer, Eli Bingham, Martin Jankowiak, Justin Chiu, Neeraj Pradhan, Alexander Rush, and Noah Goodman. Tensor variable elimination for plated factor graphs, 2019. arXiv:1902.03210.

40 Dan Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1):273–302, 1996. doi:10.1016/0004-3702(94)00092-1.

41 Steffen Smolka, Praveen Kumar, Nate Foster, Dexter Kozen, and Alexandra Silva. Cantor meets Scott: semantic foundations for probabilistic networks. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*. ACM, January 2017. doi:10.1145/3009837.3009843.

42 Steffen Smolka, Praveen Kumar, David M. Kahn, Nate Foster, Justin Hsu, Dexter Kozen, and Alexandra Silva. Scalable verification of probabilistic networks. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, June 2019. doi:10.1145/3314221.3314639.

43 Sam Staton. Commutative semantics for probabilistic programming. In Hongseok Yang, editor, *Programming Languages and Systems*, pages 855–879, Berlin, Heidelberg, 2017. Springer Berlin Heidelberg.

44 David Tolpin, Jan-Willem van de Meent, Hongseok Yang, and Frank 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, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/3064899.3064910.

45 Mateo Torres-Ruiz. Dice. https://bitbucket.org/dice-iter/dice, 2024.

46 Glynn Winskel. The formal semantics of programming languages - an introduction. In *Foundation of computing series*, 1993. URL: https://api.semanticscholar.org/CorpusID:7767429.

Table 1 Dice's core syntax. Metavariable f ranges over function names, x over variable names, and θ over reals in the range $[0, 1]$.

$$
\begin{aligned}
\tau &::= \mathbb{B} \mid \tau_1 \times \tau_2 && \text{(Types)}\\
v &::= \texttt{true} \mid \texttt{false} \mid (v, v) && \text{(Values)}\\
a &::= x \mid v && \text{(Atomic expressions)}\\
e &::= a \mid \texttt{fst}\ a \mid \texttt{snd}\ a \mid (a, a) \mid \mathsf{let}\ x = e\ \mathsf{in}\ e\\
&\quad \mid \texttt{if}\ a\ \texttt{then}\ e\ \texttt{else}\ e \mid \texttt{flip}\ \theta \mid \texttt{observe}\ a \mid f(e) && \text{(Expressions)}\\
\mathsf{func} &::= \texttt{fun}\ f(x : \tau) : \tau\ \{e\} && \text{(Functions)}\\
p &::= e \mid \texttt{func}\ p && \text{(Programs)}
\end{aligned}
$$

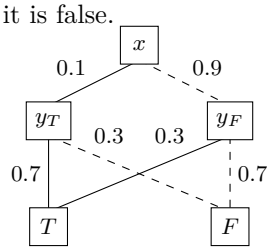
A Appendix

A.1 Core Dice

Dice programs are compiled into weighted Boolean formulas on top of which exact inference is performed via weighted model counting to obtain the distribution of densities specified in the main return expression [5]. To represent these formulas, the compiler internally manipulates programs by using multi-rooted binary decision diagrams (MRBDDs) [3], allowing it to take advantage of the conditional independence of the declared variables and the local structure amenable to efficient MRBDD compilation, making it possible for Dice to handle inference tasks even in the presence of models with a large number of variables.

Internally, the compiler maintains two Boolean formulas, φ and ψ. These represent all possible execution paths and only those that satisfy the Bayesian evidence provided by the observe statements, respectively. A global weighting function $w : L \to \mathbb{R}$ is employed to assign weights to literals. This allows for the assignment of weights to all models ω that satisfy a given formula, and the task of obtaining the posterior distribution of a given program is then reduced to a weighted model counting problem $\sum_{\omega \in \mathcal{M}(\varphi \wedge \psi)} w(\omega) / \sum_{\omega \in \mathcal{M}(\varphi)} w(\omega)$, where $\mathcal{M}(\alpha)$ is the set of models of some Boolean formula α.

▶ **Example 10.** The internal representation of the program in Example 2 is given by the following MRBDD, where the solid lines denote the paths where the decision variable is true and the dashed lines those where it is false.



where $\mathcal{M}(\varphi \wedge \psi) = \{xy_T\}$ and $\mathcal{M}(\varphi) = \{xy_T, \overline{x}y_F\}$, evaluating to $w(xy_T)/(w(xy_T) + w(\overline{x}y_F)) = (0.1 \cdot 0.7)/(0.1 \cdot 0.7 + 0.9 \cdot 0.3) \approx .2059$.

Dice can compile discrete probabilistic programs with bounded loops into a symbolic representation using MRBDDs by "unfolding" each iteration step. However, MRBDDs can only represent finite Boolean functions and thus are not suitable for directly encoding programs with unbounded loops. There is nevertheless a bijection between matrices and MRBDDs enabling us to move freely between the two representations as needed and leverage the diverse results from linear algebra to our advantage [4, 21].

▶ **Example 11.** The unfolding action on BDDs of bounded iteration leads to an increase in the number of nodes on its underlying representation, making it hard to reach sensible bounds on its approximation to the stationary distribution of the function that is iterated. Consider the problem of simulating a fair coin given samples from a biased one c with probability $p \in (0, 1)$ of landing heads. This can be done by employing von Neumann's trick: we flip c two times, if the outcomes are identical, we select the first sample as our sample, otherwise, we repeat the process. This procedure can be encode as the following program.

```
fun flip_biased_coin() {
  flip p
}

fun simulate_fair_coin(coin : bool) : (bool, bool) {
  snd (coin, coin ⊙ flip_biased_coin())
}

iterate(simulate_fair_coin, flip_biased_coin(), k)
```

While it is clear from the repeated runs of the above program that we converge to the expected value of $Bernoulli(\frac{1}{2})$, the number of nodes in its BDD representation grows linearly with the number of iterations k.

▶ Remark 12 (On normalisation). In general, the semantics of a term t might be a substochastic map. Nevertheless, eventually, the user would like to recover the normalised probability distribution encoded by a given program. While this is an expensive operation in general, the Dice compiler keeps track of an auxiliary formula, which encodes the paths through the compiled BDD allowed by the observe statements. This formula can be used to recover the normalisation constant and normalise $[\![t]\!]$ efficiently before computing $[\![\texttt{iterate}\ t]\!]$ or $\texttt{iterate}_0\ t$ from u. In other words, we always normalise before iterating. The Cesàro sum of a stochastic map is a stochastic map, so $[\![\texttt{iterate}\ t]\!]$ will also be normalised. However, this is not always the case for $\texttt{iterate}_0\ t$ from u: for example, if the the guard in $t : \tau \to \mathbb{B} \times \tau$ is always true, the semantics of the resulting iteration is uniformly zero. Note that this subprobability distribution cannot be normalised and interpreted as a genuine probability distribution. As we saw, this distribution is the bottom for the order over subprobability distributions and a program whose semantics assigns zero weights to all outcomes is better thought of as having an undefined value.

A.2 Least Fixpoint Iterator

Proof of Proposition 9. Let us call M^* the least fixpoint of φ over the poset of substochastic maps. By definition, it satisfies $M^*(\mathbf{I} - A) = B$ and $M^* \geq 0$. Furthermore, if M is any other nonnegative fixpoint of φ, we have $M^* \leq X$, so that $C^T M^* \leq C^T M$. Indeed, this holds for any pair of vectors, since C has all positive entries. Thus, M^* is indeed the solution to the linear program given in the statement. ◀

▶ Remark 13. Whenever the spectral radius of B, the transition kernel of a Markov chain representing some program $t : \tau \to \tau$, is strictly bellow 1, there is a simpler method to compute the least fixpoint of φ than the outlined in Section 5. First, recall from 5 that

$$\bigvee_{n \geq 0} \varphi^n(0) = \lim_{n \to \infty} \sum_{k=0}^{n} A \circ B^k$$

And that for T an operator $\mathbb{R}^n \to \mathbb{R}^n$, its Neumann series $\mathbf{I}+T+T^2+\dots$ converges if and only if $\rho(T) < 1$ [25]. Moreover, in this case, $(\mathbf{I}-T)$ is invertible and $\sum_{n=0}^{\infty} T^n = (\mathbf{I}-T)^{-1}$. Indeed, if $(\mathbf{I}-T)$ was not invertible, there would be $u \in \mathbb{R}^n$ different from 0 such that $Tu = u$ and hence, $\rho(T) \geq 1$ since 1 would necessarily be an eigenvalue of T.

Therefore, when $\mathsf{Ker}(\mathbf{I}-B) = \{0\}$, we have $A(\mathbf{I}-B)^{-1} = A \lim_{n\to\infty} \sum_{k=0}^{n} B^k = \lim_{n\to\infty} \sum_{k=0}^{n} AB^k$, by continuity of matrix multiplication. We see that, in this case, $A(\mathbf{I}-B)^{-1}$ is the least fixpoint of φ. Note that whenever $\mathsf{Ker}(\mathbf{I}-B)$ is not trivial, the series $\sum_{n=0}^{\infty} B^n$ diverges and this approach fails.

Impredicativity, Cumulativity and Product Covariance in the Logical Framework Dedukti

Thiago Felicissimo
Université Paris-Saclay, INRIA project Deducteam, Laboratoire Méthodes Formelles, ENS Paris-Saclay, France

Théo Winterhalter
Université Paris-Saclay, INRIA project Deducteam, Laboratoire Méthodes Formelles, ENS Paris-Saclay, France

Abstract

Proof assistants such as Coq implement a type theory featuring three important features: impredicativity, cumulativity and product covariance. This combination has proven difficult to be expressed in the logical framework Dedukti, and previous attempts have failed in providing an encoding that is proven confluent, sound and conservative. In this work we solve this longstanding open problem by providing an encoding of these three features that we prove to be confluent, sound and to satisfy a restricted (but, we argue, strong enough) form of conservativity. Our proof of confluence is a contribution by itself, and combines various criteria and proof techniques from rewriting theory. Our proof of soundness also contributes a new strategy in which the result is shown in terms of an inverse translation function, fixing a common flaw made in some previous encoding attempts.

2012 ACM Subject Classification Theory of computation → Type theory; Theory of computation → Equational logic and rewriting

Keywords and phrases Dedukti, Rewriting, Confluence, Dependent types, Cumulativity, Universes

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.21

Related Version *Full Version*: `https://hal.science/hal-04470850`

Supplementary Material *Other (Artifact with supplementary data for some proofs)*: `https://github.com/Deducteam/cc-in-dk` [19]
archived at `swh:1:dir:a3e16cdbfca697ef804c9fdac8499e773a84e8ee`

Acknowledgements The authors thank François Thiré and Yoan Géran for helpful remarks about a first draft, Gaspard Férey, Jean-Pierre Jouannaud, Frédéric Blanqui and Gilles Dowek for informative discussions around the subject of this paper, and the anonymous reviewers for their helpful comments.

1 Introduction

As the number of proof systems grow, it becomes increasingly important to understand the relationship between their logics and to which extent they can be expressed in a unified setting. The research project centered around the logical framework Dedukti [7, 16] has precisely the intent of providing such a setting. By allowing for the encoding of popular logics such as predicate logic [16], higher-order logic [32, 16], set theory [17] and pure type systems [18, 22], it provides a common framework in which proofs coming from different proof systems can be rechecked, increasing the trust in their correctness. Moreover, Dedukti can then also be used for sharing these proofs with other systems, which has already allowed for exporting results to tools like Coq [15, 44], Agda [24] and HOL [44, 29].

The correctness of the verification provided by Dedukti relies however on methatheoretic results stating that the theorems that can be proven by a Dedukti encoding are exactly the same ones of the encoded logic. In the particular case of the cumulative calculus

Table 1 Comparison with previous encodings.

	Assaf* [5]	Assaf et al. [8]	Thiré* [45]	Férey^{+} [26]	This work
Confluence	✗	✓‡	✗	✗	✓
Soundness	✗†	✗†	✗†	✓	✓
Conservativity	✗	✗	✗	✗	✓⋆

†: The translation function is ill-defined (see the discussion in Section 9).
‡: Requires matching modulo ACU. ⋆: Only in a restricted form.
∗: Also handles other cumulative type systems. +: Also supports universe polymorphism.

of constructions, a type theory combining impredicativity and cumulativity with product covariance, giving an encoding satisfying these properties has remained to this day a challenge. This issue is made especially relevant by the fact that this theory is quite popular, and is most notably implemented by the proof assistant Coq.

The current situation regarding encodings of this theory is summarised in Table 1. All encodings presented until now came with a proof of *soundness*, meaning that all facts that can be proven by the encoded logic can also be proven in the encoding. However, the proofs provided by Assaf, Assaf et al. and Thiré have turned out to be incorrect, as they rely on ill-defined translation functions – see Section 9 for a detailed explanation. The situation is even more serious regarding *conservativity*, the property dual to soundness and which ensures that the encoding cannot prove more theorems than the encoded system. Indeed, none of the previous proposals have provided a proof of this fact, which is nevertheless essential to ensure that a proof checked by Dedukti is indeed correct in the original system.

One of the challenges in proving conservativity is that all known proof methods rely on confluence – which is moreover also essential to establish subject reduction. However, the combination of impredicativity, cumulativity and product covariance has proven difficult to be expressed in a confluent way in Dedukti. Indeed, almost all previous encodings have not succeeded in proving this property. A notable exception is the impressive work of Assaf et al. [8], which however relies on matching modulo ACU (assocativity-cumutativity with unit) a form of matching that is much less efficient and harder to implement than pure syntactical matching. For instance, the addition of ACU matching to the DkCheck implementation doubled the size of the kernel [20] (see also the discussion by Blanqui [14]).

In this work we address this unsatisfying state of affairs by giving an encoding of the cumulative calculus of constructions, featuring cumulativity with product covariance, that we show to satisfy the necessary metaproperties to be used in practice.

Contrary to the previous proposals, our encoding does not require non-left-linear rewrite rules, which not only are less efficient but also make confluence proofs much harder [33]. Our proof of confluence then relies on a sophisticated combination of classical results and techniques [36, 47], and automated checkers developed by the rewriting community [30, 28, 38].

With the confluence of our encoding in hand, we proceed to show soundness. In order to fix the problem with the translation function made in previous attempts, we contribute an adaptation of the technique of Winterhalter et al. [48] and Oury [39] in which the well-typedness of the translation is stated and proved in terms of an inverse translation function. The direct translation function can then be extracted from our constructive proof of soundness.

We finish by showing that our encoding satisfies a restricted form of conservativity, namely only for so-called *object terms*. We argue that, in the encoding, these are the only terms that one writes in practice, and therefore this restricted result is sufficient.

$$\textsc{EmptyCtx}\ \frac{}{\cdot \vdash} \qquad \textsc{ExtCtx}\ \frac{\Gamma \vdash A : \mathbf{Type}}{\Gamma, x : A \vdash} \qquad \textsc{Var}\ \frac{x : A \in \Gamma \qquad \Gamma \vdash}{\Gamma \vdash x : A} \qquad \textsc{Cons}\ \frac{c : A \in \Sigma \qquad \Gamma \vdash}{\Gamma \vdash c : A}$$

$$\textsc{Sort}\ \frac{\Gamma \vdash}{\Gamma \vdash \mathbf{Type} : \mathbf{Kind}} \qquad \textsc{Conv}\ \frac{A \equiv B \qquad \Gamma \vdash t : A \qquad \Gamma \vdash B : s}{\Gamma \vdash t : B} \qquad \textsc{Pi}\ \frac{\Gamma \vdash A : \mathbf{Type} \qquad \Gamma, x : A \vdash B : s}{\Gamma \vdash (x : A) \to B : s}$$

$$\textsc{Abs}\ \frac{\Gamma \vdash A : \mathbf{Type} \qquad \Gamma, x : A \vdash B : s \qquad \Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.t : (x : A) \to B} \qquad \textsc{App}\ \frac{\Gamma \vdash t : (x : A) \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\ u : B[u/x]}$$

Figure 1 Typing rules of Dedukti.

Outline of the paper

We start in Sections 2 and 3 by recalling the definitions of Dedukti and of the variant of the calculus of constructions we consider. We then proceed in Section 4 to present the theory used in our encoding, and in Section 5 by proving its desirable properties – in particular its confluence. We define the translation function we use in Section 6, and in Sections 7 and 8 we establish the soundness and conservativity of our encoding respectively. We finish by discussing related work in Section 9, before concluding in Section 10.

2 Dedukti

We assume an underlying set $c, d, ... \in \mathcal{C}$ of *constants*, $x, y, z... \in \mathcal{V}$ of *variables* and $\mathsf{A}, \mathsf{B}, \mathsf{t}, \mathsf{u}, ... \in \mathcal{M}$ of *metavariables* equipped with an *arity* (a natural number). The *metaterms* of Dedukti [26] are defined by the following grammar.

$$\boxed{\hat{\Lambda}_{\mathsf{dk}}} \ni t, u, A, B, ... ::= x \mid c \mid \mathbf{Type} \mid \mathbf{Kind} \mid (x : A) \to B \mid \lambda x : A.t \mid t\ u \mid \mathsf{t}\{t_1, ..., t_{\mathsf{arity}(\mathsf{t})}\}$$

A *metavariable application* is written $\mathsf{t}\{t_1, ..., t_k\}$ when $\mathsf{arity}(\mathsf{t}) = k$, or just t when $\mathsf{arity}(\mathsf{t}) = 0$. The metaterms **Type** and **Kind** are called *sorts* and referred to by the letter s. We write $(x : A) \to B$ for the *dependent function type*, and whenever x does not appear free in B we write $A \to B$ instead. We define $\mathsf{fv}(t)$ as the set of free variables of t and $\mathsf{mv}(t)$ as the set of metavariables of t. When no ambiguity can arise, we allow ourselves to also write t, u, A, B for variables. We adopt the convention of writing constants names in blue font.

A *substitution* θ is a finite set of pairs t/x or $(x_1..x_k.t)/\mathsf{t}$, where $k = \mathsf{arity}(\mathsf{t})$. We write $t[\theta]$ for the application of a substitution θ to a metaterm t. The main cases of its definition are $x[\theta] = t$ when $t/x \in \theta$, and $\mathsf{t}\{u_1, ..., u_k\}[\theta] = t[u_1[\theta]/x_1, ..., u_k[\theta]/x_k]$ when $(x_1..x_k.t)/\mathsf{t} \in \theta$ – see for instance Férey [26] for the complete definition. A *rewrite system* $\mathcal{R}$ is a set of *rewrite rules*, which are pairs of the form $t \longmapsto u$ where t is of the form $c\ t_1...t_k$ and $\mathsf{fv}(t) = \mathsf{fv}(u) = \varnothing$ and $\mathsf{mv}(u) \subseteq \mathsf{mv}(t)$ and all occurrences of metavariables in t are of the form $\mathsf{t}\{x_1, ..., x_k\}$ with $x_1...x_k$ pairwise disjoint (known as the pattern condition [37]). When convenient, a rule can be given a name α, in which case we write $t \overset{\alpha}{\longmapsto} u$.

We write $\longrightarrow_{\mathcal{R}}$ for the closure under context and substitution of $\mathcal{R}$, and $\longrightarrow_{\beta\mathcal{R}}$ for $\longrightarrow_{\beta} \cup \longrightarrow_{\mathcal{R}}$ where $\longrightarrow_{\beta}$ is the usual β-reduction. Note that all these rewrite relations are defined over untyped metaterms, and that we do not consider η-reduction or -expansion, as they behave badly in the context of rewriting. We then write $\longrightarrow^*_{\beta\mathcal{R}}$ for the reflexive-transitive closure of $\longrightarrow_{\beta\mathcal{R}}$, and $\equiv_{\beta\mathcal{R}}$ for its reflexive-symmetric-transitive closure, usually called *conversion* or *definitional equality*. Most of the time $\mathcal{R}$ is clear from the context, allowing us to write just $\longrightarrow$ for $\longrightarrow_{\beta\mathcal{R}}$ and $\equiv$ for $\equiv_{\beta\mathcal{R}}$.

$$\textsc{Sub}\ \frac{n \leqslant m}{n \subseteq m} \qquad \textsc{Eq}\ \frac{A \equiv B}{A \subseteq B} \qquad \textsc{Trans}\ \frac{A \subseteq B \qquad B \subseteq C}{A \subseteq C} \qquad \textsc{ProdCov}\ \frac{A \subseteq B}{\Pi x : C.A \subseteq \Pi x : C.B}$$

$$\textsc{EmptyCtx}\ \frac{}{\cdot \vdash_{\mathsf{cc}}} \qquad \textsc{ExtCtx}\ \frac{\Gamma \vdash_{\mathsf{cc}} \qquad \Gamma \vdash_{\mathsf{cc}} A : n}{\Gamma, x : A \vdash_{\mathsf{cc}}} \qquad (x : A) \in \Gamma\ \textsc{Var}\ \frac{\Gamma \vdash_{\mathsf{cc}}}{\Gamma \vdash_{\mathsf{cc}} x : A} \qquad \textsc{Sort}\ \frac{\Gamma \vdash_{\mathsf{cc}}}{\Gamma \vdash_{\mathsf{cc}} n : \mathfrak{A}(n)}$$

$$\textsc{Pi}\ \frac{\Gamma \vdash_{\mathsf{cc}} A : n \qquad \Gamma, x : A \vdash_{\mathsf{cc}} B : m}{\Gamma \vdash_{\mathsf{cc}} \Pi x : A.B : \mathfrak{R}(n, m)} \qquad \textsc{Lam}\ \frac{\Gamma \vdash_{\mathsf{cc}} A : n \qquad \Gamma, x : A \vdash_{\mathsf{cc}} t : B}{\Gamma \vdash_{\mathsf{cc}} \lambda x : A.t : \Pi x : A.B}$$

$$\textsc{App}\ \frac{\Gamma \vdash_{\mathsf{cc}} t : \Pi x : A.B \qquad \Gamma \vdash_{\mathsf{cc}} u : A}{\Gamma \vdash_{\mathsf{cc}} t\ u : B[u/x]} \qquad A \subseteq B\ \textsc{Conv}\ \frac{\Gamma \vdash_{\mathsf{cc}} t : A \qquad \Gamma \vdash_{\mathsf{cc}} B : n}{\Gamma \vdash_{\mathsf{cc}} t : B}$$

Figure 2 Typing rules for CC.

Metavariables are useful in order to define the notion of rewrite rules, but apart from this they will have no use for us, and in particular typing will only be defined for metaterms without metavariables. Because of this, we define the set of Dedukti *terms* Λ_{dk} as the metaterms t satisfying $\mathsf{mv}(t) = \varnothing$. Given that terms will be the main object of study, from now on we adopt the convention that the letters $t, u, A, B, \ldots$ refer to terms, unless it is explicitly said that they refer to metaterms.

A *context* Γ is a finite sequence of entries of the form $x : A$. A *signature* Σ is a (possibly infinite) sequence of entries of the form $c : A$. One central notion in Dedukti is that of *theory*, which is a pair $\mathbb{T} = (\Sigma_{\mathbb{T}}, \mathcal{R}_{\mathbb{T}})$ where $\Sigma_{\mathbb{T}}$ is a signature and all constants appearing in $\mathcal{R}_{\mathbb{T}}$ are declared in $\Sigma_{\mathbb{T}}$. Theories are used in Dedukti to define the object logics in which we work (for instance, predicate logic). Given a theory $\mathbb{T}$, the typing rules of Dedukti are given in Figure 1, where the signature Σ and the conversion relation $\equiv$ are the ones defined by the theory $\mathbb{T}$. Whenever $\mathbb{T}$ is not clear from the context, we write $\mathbb{T} \triangleright \Gamma \vdash t : A$.

A signature entry $c : A$ is valid in $\mathbb{T}$ when $\mathbb{T} \triangleright \cdot \vdash A : s$ for some sort s. A theory $\mathbb{T}$ is said to be *well typed* when each entry $c : A \in \Sigma_{\mathbb{T}}$ is valid in $(\Sigma', \mathcal{R}')$, where Σ' is the prefix of $\Sigma_{\mathbb{T}}$ preceding $c : A$, and $\mathcal{R}'$ is the restriction of $\mathcal{R}_{\mathbb{T}}$ to rules only containing constants in Σ'.

3 The Cumulative Calculus of Constructions with Product Covariance

We recall the definition of the cumulative calculus of constructions with product covariance [35, 31]. It can be seen as the underlying *cumulative type system* [34, 10] of the Coq proof assistant [42], omitting the sorts Set and SProp. Its syntax is given by the following grammar.

$$\boxed{\Lambda_{\mathsf{cc}}} \ni \qquad t, u, A, B ::= x \mid n \mid \Pi x : A.B \mid \lambda x : A.t \mid t\ u$$

Here we have made the choice of representing universes directly by a natural number n. The universe that is commonly referred to as Prop then corresponds to 0, whereas Type_n corresponds to $n + 1$, allowing us to manipulate them in a more uniform way. The typing rules are then given in Figure 2, and are parametrized by the following *axiom* and *rule* functions, as they are known in the *pure type system* literature [27, Definition 4.3.2].

$$\begin{aligned}
&\mathfrak{A} : \mathbb{N} \to \mathbb{N} && \mathfrak{R} : \mathbb{N} \times \mathbb{N} \to \mathbb{N} \\
&\mathfrak{A}(0) := 2 && \mathfrak{R}(n, 0) := 0 \\
&\mathfrak{A}(1+n) := 2 + n && \mathfrak{R}(n, 1+m) := \mathsf{max}\{n, 1+m\}
\end{aligned}$$

▶ Remark 1. We choose to follow the implementation of Coq in placing 0 (Prop) in the universe 2 (Type_1). Some presentations choose instead to place it in 1 (Type_0) [35], a technical change that would have no impact in the strategy developed in this paper.

Compared with type systems that do not feature cumulativity, the conversion rule for CC does not only allow to exchange two types A and B when they are convertible, but also to coerce a term from type A to B when the former is a subtype of the latter. This subtyping relation, written $A \subseteq B$, is defined in the base case as $A \subseteq B$ when $A \equiv B$, or $n \subseteq m$ when $n \leqslant m$. The second rule allows us for instance to coerce a type $\Gamma \vdash A : 0$ to $\Gamma \vdash A : 1$. Then, what one calls *product covariance* is the rule allowing to deduce $\Pi x : C.A \subseteq \Pi x : C.B$ from $A \subseteq B$, which lets us for instance to coerce a function $\Gamma \vdash_{\mathsf{cc}} f : \mathsf{Nat} \to 0$ to $\Gamma \vdash_{\mathsf{cc}} f : \mathsf{Nat} \to 1$.

4 Introducing the theory $\mathbb{T}_{\mathsf{cc}}$

We now introduce the Dedukti theory $\mathbb{T}_{\mathsf{cc}}$ we will use in our encoding. We build it incrementally in order to motivate as best as possible the choices we have made.

Our first step is declaring a type $\mathfrak{S}$ along with constants 0 and S for zero and successor, allowing us to represent the CC sort n by the Dedukti term $\mathsf{S}^n\ 0$ – which from now on we write as $\underline{n}$. We then define many auxiliary constants that will be useful later, such as addition $+$, truncated predecessor P, and also constants $\mathfrak{A}$ and $\mathfrak{R}$ to represent the functions $\mathfrak{A}$ and $\mathfrak{R}$ from the definition of CC. We declare the associated rewrite rules so that they have the expected computational behavior, such as $\underline{n} + \underline{m} \longrightarrow^* \underline{n+m}$, $\underline{n} \vee \underline{m} \longrightarrow^* \underline{\mathsf{max}\{n, m\}}$, etc.

$$\begin{array}{llll}
\mathfrak{S} : \mathbf{Type} & \mathfrak{A} : \mathfrak{S} \to \mathfrak{S} & \mathsf{P} : \mathfrak{S} \to \mathfrak{S} & - : \mathfrak{S} \to \mathfrak{S} \to \mathfrak{S} \quad \text{(infix)} \\
0 : \mathfrak{S} & \mathfrak{A}\ 0 \longmapsto \mathsf{S}\ (\mathsf{S}\ 0) & \mathsf{P}\ 0 \longmapsto 0 & \mathsf{l}_1 - 0 \longmapsto \mathsf{l}_1 \\
\mathsf{S} : \mathfrak{S} \to \mathfrak{S} & \mathfrak{A}\ (\mathsf{S}\ \mathsf{l}) \longmapsto \mathsf{S}\ (\mathsf{S}\ \mathsf{l}) & \mathsf{P}\ (\mathsf{S}\ \mathsf{l}) \longmapsto \mathsf{l} & \mathsf{l}_1 - (\mathsf{S}\ \mathsf{l}_2) \longmapsto (\mathsf{P}\ \mathsf{l}_1) - \mathsf{l}_2
\end{array}$$

$$\begin{array}{lll}
+ : \mathfrak{S} \to \mathfrak{S} \to \mathfrak{S} \quad \text{(infix)} & \vee : \mathfrak{S} \to \mathfrak{S} \to \mathfrak{S} \quad \text{(infix)} & \mathfrak{R} : \mathfrak{S} \to \mathfrak{S} \to \mathfrak{S} \\
0 + \mathsf{l}_2 \longmapsto \mathsf{l}_2 & 0 \vee \mathsf{l}_2 \longmapsto \mathsf{l}_2 & \mathfrak{R}\ \mathsf{l}_1\ 0 \longmapsto 0 \\
\mathsf{l}_1 + 0 \longmapsto \mathsf{l}_1 & \mathsf{l}_1 \vee 0 \longmapsto \mathsf{l}_1 & \mathfrak{R}\ \mathsf{l}_1\ (\mathsf{S}\ \mathsf{l}_2) \longmapsto \mathsf{l}_1 \vee (\mathsf{S}\ \mathsf{l}_2) \\
(\mathsf{S}\ \mathsf{l}_1) + \mathsf{l}_2 \longmapsto \mathsf{S}\ (\mathsf{l}_1 + \mathsf{l}_2) & (\mathsf{S}\ \mathsf{l}_1) \vee (\mathsf{S}\ \mathsf{l}_2) \longmapsto \mathsf{S}\ (\mathsf{l}_1 \vee \mathsf{l}_2) & \\
\mathsf{l}_1 + (\mathsf{S}\ \mathsf{l}_2) \longmapsto \mathsf{S}\ (\mathsf{l}_1 + \mathsf{l}_2) & &
\end{array}$$

Using $\mathfrak{S}$ we can then encode the universes of CC. This is done by declaring a constant U, such that the inhabitants of $\mathsf{U}_{\underline{n}}$ can then be thought of as codes for the types of CC in n. The decoding function El then maps each such code to the Dedukti type of its elements.

$$\mathsf{U} : (l : \mathfrak{S}) \to \mathbf{Type} \quad \text{(written } \mathsf{U}_l\text{)} \qquad \mathsf{El} : (l : \mathfrak{S}) \to \mathsf{U}_l \to \mathbf{Type} \quad \text{(written } \mathsf{El}_l\text{)}$$

Next we add constants to represent the codes inhabiting such universes. Because in CC each universe n inhabits $\mathfrak{A}(n)$, we add a constant u mapping each $l : \mathfrak{S}$ to its code in $\mathsf{U}_{(\mathfrak{A}\ l)}$. An associated rewrite rule then ensures that u_l decodes to the type U_l as expected.

$$\mathtt{u} : (l : \mathfrak{S}) \to \mathrm{U}_{(\mathfrak{R}\ l)} \quad \text{(written } \mathrm{u}_l\text{)} \qquad\qquad \mathrm{El}_{(\_)}\ \mathrm{u}_1 \overset{\mathrm{El_u}}{\longmapsto} \mathrm{U}_1$$

A similar story happens for the function type: we add a constant π mapping a code $a : \mathrm{U}_{l_a}$ and a family of codes $b : \mathrm{El}_{l_a}\ a \to \mathrm{U}_{l_b}$ to a code in $\mathrm{U}_{(\mathfrak{R}\ l_a\ l_b)}$, so that if a represents A and b represents B, then the result represents the CC type $\Pi x : A.B$. However, for reasons that will become clear later, our constant also allows us to decompose the sorts l_a and l_b into a common factor l_0 to which we apply offsets l_1 and l_2. In order to equate different decompositions of l_a and l_b, we also add a rewrite rule which removes two successors of l_1 and l_2 and compensates it by adding one in l_0. Finally, we add a rewrite rule defining the elements of $\pi^{l_0}_{l_1,l_2}\ a\ \lambda x.b$ as the Dedukti functions from the elements of a to the ones of b.

$$\begin{aligned}
&\pi : (l_0\ l_1\ l_2 : \mathfrak{S}) \to (A : \mathrm{U}_{(l_0+l_1)}) \\
&\qquad \to (B : \mathrm{El}_{(l_0+l_1)}\ A \to \mathrm{U}_{(l_0+l_2)}) \to \mathrm{U}_{(\mathfrak{R}\ (l_0+l_1)\ (l_0+l_2))} \qquad \text{(written } \pi^{l_0}_{l_1,l_2}\text{)} \\
&\pi^{l_0}_{(\mathrm{S}\ l_1),(\mathrm{S}\ l_2)}\ \mathtt{A}\ \mathtt{B} \overset{\pi_\mathrm{S}}{\longmapsto} \pi^{(\mathrm{S}\ l_0)}_{l_1,l_2}\ \mathtt{A}\ \mathtt{B} \\
&\mathrm{El}_{(_)}\ (\pi^{l_0}_{l_1,l_2}\ \mathtt{A}\ \lambda x : \mathtt{C.B}\{x\}) \overset{\mathrm{El}_\pi}{\longmapsto} (x : \mathrm{El}_{(l_0+l_1)}\ \mathtt{A}) \to \mathrm{El}_{(l_0+l_2)}\ \mathtt{B}\{x\}
\end{aligned}$$

The theory given until this point is a representation of CC *without* cumulativity, and straightforwardly applies well-known techniques from previous Dedukti encodings [18, 16]. The interesting part is for the encoding of cumulativity. The main insight of our proposal comes from the following simple result regarding the relation $\subseteq$. In the following, given a context $\Delta = x_1 : B_1 .. x_k : B_k$, let us write $\Delta \Rightarrow A$ for the CC term $\Pi x_1 : B_1 ... x_k : B_k.A$.

▶ **Lemma 2** (Case analysis of $\subseteq$). *If $A \subseteq B$ then either $A \equiv B$ or $A \longrightarrow^* \Delta \Rightarrow n$ and $B \longrightarrow^* \Delta \Rightarrow m$ for some context Δ and natural numbers n, m with $n \leqslant m$.*

Therefore, in order to simulate CC's cumulativity it suffices to add a *lift* $\uparrow$ allowing the coercion of terms from a type $\Delta \Rightarrow n$ to $\Delta \Rightarrow n+1$. However, to be able to state the type of $\uparrow$ we first need to have an internal representation for types of the form $\Delta \Rightarrow n$ in Dedukti. We do this by first defining a type for *telescopes* whose canonical elements are either the empty telescope $\blacklozenge$, or the extension $A\ l \blacktriangleleft \lambda x.D$ of a telescope D with a code A in universe U_l. We can then define a function $\Rightarrow$ that computes a Dedukti type corresponding to $\Delta \Rightarrow n$.

$$\begin{aligned}
&\mathrm{Tele} : \mathbf{Type} && \Rightarrow : \mathrm{Tele} \to \mathfrak{S} \to \mathbf{Type} \quad \text{(infix)} \\
&\blacklozenge : \mathrm{Tele} && \blacklozenge \Rightarrow \mathtt{l}_1 \overset{\Rightarrow_\blacklozenge}{\longmapsto} \mathrm{U}_{\mathtt{l}_1} \\
&\blacktriangleleft : (l : \mathfrak{S}) \to (A : \mathrm{U}_l) \to (\mathrm{El}_l\ A \to \mathrm{Tele}) && (\mathtt{A}\ _{\mathtt{l}_2} \blacktriangleleft \lambda x : _.\mathtt{D}\{x\}) \Rightarrow \mathtt{l}_1 \overset{\Rightarrow_\blacktriangleleft}{\longmapsto} (x : \mathrm{El}_{\mathtt{l}_2}\ \mathtt{A}) \to \mathtt{D}\{x\} \Rightarrow \mathtt{l}_1 \\
&\quad \to \mathrm{Tele} \quad \text{(infix, written } {}_l\blacktriangleleft\text{)}
\end{aligned}$$

With these definitions in place we can finally give the definition of $\uparrow$.[1]

$$\uparrow : (l : \mathfrak{S}) \to (D : \mathrm{Tele}) \to (D \Rightarrow l) \to (D \Rightarrow (\mathrm{S}\ l)) \quad \text{(written } \uparrow_l\text{)}$$

Because in CC the applications of cumulativity are silent, the main challenge in the encoding is to ensure that different Dedukti representations of the same CC term are convertible. The pioneering work of Assaf [4] first identified that, in a setting without product covariance, it suffices to add the following *full reflection* equations – here and in the rest of the article we write $\uparrow^m_{\underline{n}}\ D\ t$ as a notation for $\uparrow_{\underline{m-1}}\ D\ (...(\uparrow_{\underline{n}}\ D\ t)...)$ when $n \leqslant m$.

[1] Note that our lift is *single-step*, in contrast with some previous encodings [5, 45, 26] which employed a *multi-step* lift, taking a type $A : \mathrm{U}_{l_1}$ to $\uparrow^{l_2}_{l_1} \blacklozenge\ t : \mathrm{U}_{l_2}$. The avoidance of the multi-step lift is essential in order to prevent its associated non-left-linear rules, such as $\uparrow^1_1 \mathtt{D}\ \mathtt{t} \longmapsto \mathtt{t}$.

$$\pi^{0}_{\underline{1+n},\underline{m}}\ (\uparrow_{\underline{n}} \blacklozenge a)\ (\lambda x.b) \equiv \uparrow^{\frac{\mathfrak{R}(1+n,m)}{\mathfrak{R}(n,m)}} \blacklozenge (\pi^{0}_{\underline{n},\underline{m}}\ a\ (\lambda x.b))$$
$$\pi^{0}_{\underline{n},\underline{1+m}}\ a\ (\lambda x.\uparrow_{\underline{m}} \blacklozenge b) \equiv \uparrow^{\frac{\mathfrak{R}(n,1+m)}{\mathfrak{R}(n,m)}} \blacklozenge (\pi^{0}_{\underline{n},\underline{m}}\ a\ (\lambda x.b))$$

The main difficulty in implementing these as rewrite rules is that the multistep lift $\uparrow^{\underline{m}}_{\underline{n}}$ is just a notation which computes the correct number of lifts $\uparrow$ to be inserted only for a given concrete choice of n and m. For instance, if $n > m > 0$ in the second equation then no lifts should be inserted in the right hand side, whereas if $n > m = 0$ then we must insert $n-1$ lifts. Of course, this could be solved by adding these as infinite schemes of rewrite rules, for all $n, m \in \mathbb{N}$, however we want to keep the rewrite system finitary, so it can actually be used in practice in the implementation.

If only we could have more information about n and m when applying the rule, we would be able to calculate the correct amount of lifts. Thankfully, because the sorts of a and b can be decomposed with the rule π_{S}, we know that for any $\pi^{n_0}_{\underline{n_1},\underline{n_2}}\ a\ \lambda x.b$ in normal form we must have either $n_1 = 0$ or $n_2 = 0$. We can then proceed with a disjunction of cases, where in each situation we have enough information to apply the right number of lifts.

$$\pi^{0}_{(\mathrm{S}\ 1),0}\ (\uparrow_{\_} \blacklozenge \mathsf{A})\ \mathsf{B} \xmapsto{\uparrow^1_\pi} \pi^{0}_{1,0}\ \mathsf{A}\ \mathsf{B}$$
$$\pi^{(\mathrm{S}\ 1_1)}_{0,1_2}\ (\uparrow_{\_} \blacklozenge \mathsf{A})\ \mathsf{B} \xmapsto{\uparrow^2_\pi} \pi^{1_1}_{0,(\mathrm{S}\ 1_2)}\ \mathsf{A}\ \mathsf{B}$$
$$\pi^{(\mathrm{S}\ 1_1)}_{(\mathrm{S}\ 1_2),0}\ (\uparrow_{\_} \blacklozenge \mathsf{A})\ \mathsf{B} \xmapsto{\uparrow^3_\pi} \uparrow_{(\mathrm{S}\ (1_1+1_2))} \blacklozenge (\pi^{(\mathrm{S}\ 1_1)}_{1_2,0}\ \mathsf{A}\ \mathsf{B})$$

$$\Uparrow : (l : \mathfrak{S}) \to (A : \mathrm{U}_0) \to \mathrm{U}_l \quad \text{(written } \Uparrow_l\text{)}$$
$$\Uparrow_0\ \mathsf{A} \longmapsto \mathsf{A}$$
$$\Uparrow_{(\mathrm{S}\ 1)}\ \mathsf{A} \longmapsto \uparrow_1 \blacklozenge (\Uparrow_1\ \mathsf{A})$$

$$\pi^{(\mathrm{S}\ (\mathrm{S}\ 1_1))}_{1_2,0}\ \mathsf{A}\ (\lambda x : \mathsf{C}.\uparrow_{\_} \blacklozenge \mathsf{B}\{x\}) \xmapsto{\uparrow^4_\pi} \pi^{(\mathrm{S}\ 1_1)}_{(\mathrm{S}\ 1_2),0}\ \mathsf{A}\ (\lambda x : \mathsf{C}.\mathsf{B}\{x\})$$
$$\pi^{1_1}_{0,(\mathrm{S}\ 1_2)}\ \mathsf{A}\ (\lambda x : \mathsf{C}.\uparrow_{\_} \blacklozenge \mathsf{B}\{x\}) \xmapsto{\uparrow^5_\pi} \uparrow_{(1_1+1_2)} \blacklozenge (\pi^{1_1}_{0,1_2}\ \mathsf{A}\ (\lambda x : \mathsf{C}.\mathsf{B}\{x\}))$$
$$\pi^{(\mathrm{S}\ 0)}_{1,0}\ \mathsf{A}\ (\lambda x : \mathsf{C}.\uparrow_{\_} \blacklozenge \mathsf{B}\{x\}) \xmapsto{\uparrow^6_\pi} \Uparrow_{(\mathrm{S}\ 1)}\ (\pi^{0}_{(\mathrm{S}\ 1),0}\ \mathsf{A}\ (\lambda x : \mathsf{C}.\mathsf{B}\{x\}))$$

Note that in order to state the last rule we also define an auxiliary constant $\Uparrow$ which given a sort l, lifts a type from U_0 to U_l. The following proposition then ensures that we have correctly implemented Assaf's full reflection equations.

▶ **Proposition 3** (Simulation of Assaf's full reflection rules). *We have the following conversions.*

$$\pi^{0}_{\underline{1+n},\underline{m}}\ (\uparrow_l \blacklozenge a)\ (\lambda x : C.b) \equiv \uparrow^{\frac{\mathfrak{R}(1+n,m)}{\mathfrak{R}(n,m)}} \blacklozenge (\pi^{0}_{\underline{n},\underline{m}}\ a\ (\lambda x : C.b)) \tag{1}$$
$$\pi^{0}_{\underline{n},\underline{1+m}}\ a\ (\lambda x : C.\uparrow_l \blacklozenge b) \equiv \uparrow^{\frac{\mathfrak{R}(n,1+m)}{\mathfrak{R}(n,m)}} \blacklozenge (\pi^{0}_{\underline{n},\underline{m}}\ a\ (\lambda x : C.b)) \tag{2}$$

Proof. By a disjunction of cases in which each case corresponds to one of the rules $\uparrow^i_\pi$. ◀

▶ Remark 4. We note that the rules $\uparrow^i_\pi$ are also very similar to the ones identified by Assaf et al. [8]. However they also differ in a crucial way by avoiding the use of non-left-linearity and matching modulo ACU, which are less efficient and render confluence proofs much harder.

The rules given until now would ensure the uniqueness of codes for a version of CC with "simple" cumulativity. However, in a setting with product covariance we also need to ensure that $\uparrow$ properly commutes with abstraction and application. We therefore add the following two rules, which are variants of similar equations first identified by Thiré [45] and Férey [26].

$$\uparrow_1\ (\_\ (\_)\ \blacktriangleleft\ \lambda x : \_.\mathtt{D}\{x\})\ \lambda x : \mathtt{A}.\mathtt{t}\{x\} \overset{\uparrow_\lambda}{\longmapsto} \lambda x : \mathtt{A}.\uparrow_1\ \mathtt{D}\{x\}\ \mathtt{t}\{x\}$$
$$\uparrow_1\ (\_\ (\_)\ \blacktriangleleft\ \lambda x : \_.\mathtt{D}\{x\})\ \mathtt{t}\ \mathtt{u} \overset{\uparrow_@}{\longmapsto} \uparrow_1\ \mathtt{D}\{\mathtt{u}\}\ (\mathtt{t}\ \mathtt{u})$$

We now have almost finished presenting the theory $\mathbb{T}_{\mathsf{cc}}$. The final step is adding the following rule explaining the relationship between the elements of $\uparrow_l \blacklozenge A$ and the ones of A, which as expected should be the same. Here we have purposely avoided the expected rule $\mathrm{El}_{(\mathtt{S}\ 1)}\ (\uparrow_{(\_)} \blacklozenge \mathtt{A}) \longmapsto \mathrm{El}_1\ \mathtt{A}$ used in some previous proposals [5, 45]. This subtle difference is essential in order to allow the critical pairs between $\uparrow^i_\pi$ and El_π to close. We add a similar rule for $\Uparrow$, but once again we annotate El with $\mathtt{l}_2 - \mathtt{l}_1$ instead of 0 in order to ensure that critical pairs all close. Finally, we need a last rule similar to $\uparrow_{\mathrm{El}}$ ensuring the uniqueness of telescope representations, which will be key when proving the injectivity of $\Rightarrow$.

$$\mathrm{El}_1\ (\uparrow_{\_} \blacklozenge \mathtt{A}) \overset{\uparrow_{\mathrm{El}}}{\longmapsto} \mathrm{El}_{(\mathtt{P}\ 1)}\ \mathtt{A} \qquad \mathrm{El}_{\mathtt{l}_2}\ (\Uparrow_{\mathtt{l}_1}\ \mathtt{A}) \overset{\Uparrow_{\mathrm{El}}}{\longmapsto} \mathrm{El}_{(\mathtt{l}_2 - \mathtt{l}_1)}\ \mathtt{A} \qquad (\uparrow_{\_} \blacklozenge \mathtt{A})_1 \blacktriangleleft \mathtt{D} \overset{\uparrow_\blacktriangleleft}{\longmapsto} \mathtt{A}_{(\mathtt{P}\ 1)} \blacktriangleleft \mathtt{D}$$

5 Basic properties of $\mathbb{T}_{\mathsf{cc}}$

With the definition of the theory $\mathbb{T}_{\mathsf{cc}}$ in place, we now show that it satisfies the basic properties one expects, which will be essential for proving soundness and conservativity later. The first of them is the fact the the theory $\mathbb{T}_{\mathsf{cc}}$ is well-typed, in the sense defined in Section 2.

▶ **Proposition 5** (Well-typedness of $\mathbb{T}_{\mathsf{cc}}$). *The theory $\mathbb{T}_{\mathsf{cc}}$ is well typed.*

Proof. Checked automatically with Lambdapi – see the artifact [19] for more details. ◀

5.1 Confluence

Unlike all previous proposals, our theory $\mathbb{T}_{\mathsf{cc}}$ only makes use of left-linear rules. By preventing the use of non-left-linearity, which interacts very badly with higher-order rewriting, we have made a first step for proving confluence. Yet, confluence still does not come for free. In order to show it, we split $\beta\mathcal{R}_{\mathsf{cc}}$ into subsystems $\beta\mathcal{R}_1$ and $\mathcal{R}_2$, allowing us to apply different techniques for showing their confluence. Note that the union $\beta\mathcal{R}_1 \cup \mathcal{R}_2$ is not disjoint: the rule $\uparrow_{\mathrm{El}}$, needed for closing critical pairs in both subsystems, is shared between them.

$$\mathcal{R}_1 := \{\uparrow_@, \uparrow_\lambda, \uparrow_\blacktriangleleft, \Rightarrow_\blacklozenge, \Rightarrow_\blacktriangleleft, \uparrow_{\mathrm{El}}\} \qquad \mathcal{R}_2 := \mathcal{R}_{\mathsf{cc}} \backslash \{\uparrow_@, \uparrow_\lambda, \uparrow_\blacktriangleleft, \Rightarrow_\blacklozenge, \Rightarrow_\blacktriangleleft\}$$

Confluence of $\beta\mathcal{R}_1$

The hardest part of our proof is showing the confluence of $\beta\mathcal{R}_1$, for two main reasons. First, even though all critical pairs of $\beta\mathcal{R}_1$ close (as shown in Figure 3), because the β rule is non-normalizing on untyped terms, we cannot apply Newman's Lemma to reduce proving confluence to local confluence. Second, because the critical pairs are neither *trivial* [47] nor *development closed* [46], we cannot apply the classical criteria that avoid the use of termination. Thankfully, it turns out that we can still employ the well-known technique of showing that *orthogonal rewriting* with $\beta\mathcal{R}_1$ satisfies the diamond property, from which confluence of $\beta\mathcal{R}_1$ will follow as a simple corollary.

Given a rewrite system $\mathcal{R}$, the *orthogonal rewriting relation* $\Longrightarrow_{\beta\mathcal{R}}$ [21, 26] (also known as *developments* or *multi-step reduction* [11]) is defined over metaterms by the following inference rules, where we write $\theta \Longrightarrow \theta'$ as an abbreviation for $\mathsf{dom}(\theta) = \mathsf{dom}(\theta')$ and for all

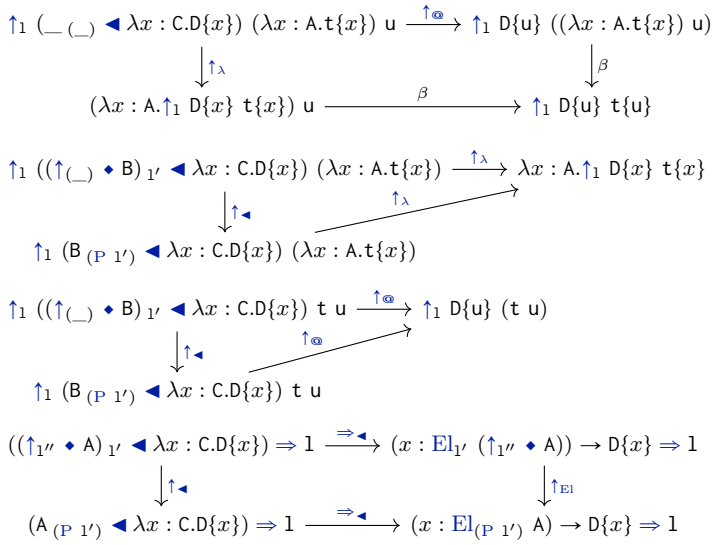


Figure 3 Critical pairs of $\beta\mathcal{R}_1$.

$\vec{x}.t/\mathtt{t} \in \theta$ and $\vec{x}.t'/\mathtt{t} \in \theta'$ we have $t \Longrightarrow t'$.

$$\textsc{Var}\ \frac{}{x \Longrightarrow x} \qquad \textsc{Const}\ \frac{}{c \Longrightarrow c} \qquad \textsc{Sort}\ \frac{}{s \Longrightarrow s} \qquad \textsc{Meta}\ \frac{t_i \Longrightarrow t'_i \quad \text{for all } i}{\mathtt{t}\{t_1..t_k\} \Longrightarrow \mathtt{t}\{t'_1..t'_k\}}$$

$$\textsc{App}\ \frac{t \Longrightarrow t' \qquad u \Longrightarrow u'}{t\ u \Longrightarrow t'\ u'} \qquad \textsc{Abs}\ \frac{A \Longrightarrow A' \qquad t \Longrightarrow t'}{\lambda x : A.t \Longrightarrow \lambda x : A'.t'} \qquad \textsc{Fun}\ \frac{A \Longrightarrow A' \qquad B \Longrightarrow B'}{(x : A) \to B \Longrightarrow (x : A') \Longrightarrow B'}$$

$$\textsc{Red}_{\mathcal{R}}\ \frac{l \longmapsto r \in \mathcal{R} \qquad \theta \Longrightarrow \theta'}{\mathsf{mv}(l) = \mathsf{dom}(\theta)\ \ l[\theta] \Longrightarrow r[\theta']} \qquad \textsc{Red}_{\beta}\ \frac{t \Longrightarrow t' \qquad u \Longrightarrow u'}{(\lambda x : A.t)\ u \Longrightarrow t'[u'/x]}$$

For all $\mathcal{R}$, orthogonal rewriting satisfies the following well-known properties – see for instance [11].

▶ **Proposition 6.** *We have* $\longrightarrow_{\beta\mathcal{R}} \subseteq \Longrightarrow_{\beta\mathcal{R}} \subseteq \longrightarrow^*_{\beta\mathcal{R}}$*, hence* $\longrightarrow^*_{\beta\mathcal{R}}$ *and* $\Longrightarrow^*_{\beta\mathcal{R}}$ *are equal.*

▶ **Proposition 7.** *If* $t \Longrightarrow_{\beta\mathcal{R}} t'$ *and* $\theta \Longrightarrow_{\beta\mathcal{R}} \theta'$ *then* $t[\theta] \Longrightarrow_{\beta\mathcal{R}} t'[\theta']$.

▶ **Proposition 8.** *We have* $t \Longrightarrow_{\beta\mathcal{R}} t$ *for all* t.

Using these properties, we can now show the following:

▶ **Proposition 9.** $\Longrightarrow_{\beta\mathcal{R}_1}$ *satisfies the diamond property.*

Proof. Given t, u, v with $u \Longleftarrow t \Longrightarrow v$ we show that there is w with $u \Longrightarrow w \Longleftarrow v$. The proof is by induction on $t \Longrightarrow u$ and $t \Longrightarrow v$. The only interesting case is when $t \Longrightarrow u$ (or dually, $t \Longrightarrow v$) is derived with rules $\textsc{Red}_{\mathcal{R}}$ or $\textsc{Red}_{\beta}$. The case $\textsc{Red}_{\beta}$ follows by the same argument as in the proof of confluence for the λ-calculus [9, Lemma 3.2.6], so let us now consider the case $\textsc{Red}_{\mathcal{R}}$, in which we have $t = l[\theta]$ for some $l \longmapsto r \in \mathcal{R}_1$ and $u = r[\theta']$ with $\theta \Longrightarrow \theta'$. There are then three possibilities regarding $t \Longrightarrow v$.

- If all applications of $\textsc{Red}_{\mathcal{R}}$ or $\textsc{Red}_{\beta}$ in $t \Longrightarrow v$ occur inside the substitution θ, then because l is linear we have $v = l[\theta'']$ with $\theta \Longrightarrow \theta''$. By i.h. we have $\theta' \Longrightarrow \theta''' \Longleftarrow \theta''$ for some θ''', and thus $u = r[\theta'] \Longrightarrow r[\theta'''] \Longleftarrow l[\theta''] = v$ using Propositions 7 and 8.
- If $t \Longrightarrow v$ starts with an application of $\textsc{Red}_{\mathcal{R}}$ using the same rule as the one applied in $t \Longrightarrow u$, then we have $v = r[\theta'']$ with $\theta \Longrightarrow \theta''$. By i.h. we have $\theta' \Longrightarrow \theta''' \Longleftarrow \theta''$ for some θ''', and thus $u = r[\theta'] \Longrightarrow r[\theta'''] \Longleftarrow r[\theta''] = v$ using Propositions 7 and 8.
- If in $t \Longrightarrow v$ there is at least one application of $\textsc{Red}_{\mathcal{R}}$ with a rule $l' \longmapsto r'$ destroying the redex $t = l[\theta]$ (note that this is impossible with $\textsc{Red}_{\beta}$), we consider all such possible cases. In our case, this turns out to correspond exactly to the critical pairs[2] in Figure 3. We can then conclude each of these cases by appealing to the i.h. and Proposition 7 to close the diagrams. The following diagram illustrates this for when $t \Longrightarrow u$ is derived with $\uparrow_{@}$ and $t \Longrightarrow v$ destroys its redex with $\uparrow_{\lambda}$. In the middle square, X stands for C, D, A, t, u.

$$\begin{array}{ccc}
\uparrow_l\ (_\ (_) \blacktriangleleft \lambda x : C.D)\ (\lambda x : A.t)\ u & \Longrightarrow & \uparrow_{l'}\ D'[u'/x]\ ((\lambda x : A'.t')\ u') \\
\Big\Vert & & \vdots \\
 & \begin{array}{ccc} X & \Longrightarrow & X' \\ \Downarrow & i.h. & \vdots \\ X'' & \Rightarrow & X''' \end{array} & \\
(\lambda x : A''.\uparrow_{l''} D''\ t'')\ u'' & \Rightarrow & \uparrow_{l'''} D'''[u'''/x]\ t'''[u'''/x] \blacktriangleleft
\end{array}$$

Combining Proposition 9 with Proposition 6, we immediately get the following corollary.

▶ **Corollary 10.** *$\beta\mathcal{R}_1$ is confluent.*

▶ Remark 11. Alternatively, one can show the confluence of $\beta\mathcal{R}_1$ by applying a recent criterion by Dowek, Férey, Jouannaud and Liu [21, Theorem 38]. However, the proof we give is more elementary as it relies neither on orthogonal critical pairs nor on decreasing diagrams, and therefore we believe that it is accessible to a wider audience.

Confluence of $\mathcal{R}_2$

We now move to the proof of confluence of $\mathcal{R}_2$, which relies on termination.

▶ **Lemma 12.** *$\mathcal{R}_2$ is strongly normalizing.*

Proof. We translate from $\mathcal{R}_2$ into the first-order rewrite system $\hat{\mathcal{R}}_2$ obtained by forgetting about binders: $\lambda x : A.t$ is translated into $\hat{\lambda}\ A'\ t'$ and $\Pi x : A.B$ is translated into $\hat{\Pi}\ A'\ B'$, where A', B', t' are the translations of A, B, t. For instance, the rule $\uparrow^4_{\pi}$ is translated into the rule $\pi^{(\mathsf{S}\ (\mathsf{S}\ 1_1))}_{1_2,0}\ \mathsf{A}\ (\hat{\lambda}\ \mathsf{C}\ (\uparrow\_ \blacklozenge \mathsf{B})) \longmapsto \pi^{(\mathsf{S}\ 1_1)}_{(\mathsf{S}\ 1_2),0}\ \mathsf{A}\ (\hat{\lambda}\ \mathsf{C}\ \mathsf{B})$. We can easily show that this interpretation preserves reduction sequences, therefore we reduce SN of $\mathcal{R}_2$ to the one of $\hat{\mathcal{R}}_2$. The latter can be shown with the use of the first-order termination checker AProVE [1, 28], and the proof can be verified by the formally certified tool CeTA [2, 43] – see the artifact [19]. ◀

▶ **Proposition 13.** *$\mathcal{R}_2$ is confluent.*

Proof. We use the tools CSI$^{\mathsf{ho}}$ [3, 38] and SOL [30] to verify that all critical pairs of $\mathcal{R}_2$ are joinable – see the artifact [19] for details – so by Mayr and Nipkow's critical pair criterion [36, Theorem 4.7] we conclude that $\mathcal{R}_2$ is locally confluent.[3] Together with Lemma 12, this gives the confluence of $\mathcal{R}_2$ by applying Newman's Lemma. ◀

[2] Although, for arbitrary $\mathcal{R}$, it is not true in general that all such situations arise from simple critical pairs, and one needs instead to consider the more general notion of *orthogonal* critical pairs [21].

[3] Note that, although Mayr and Nipkow's criterion was shown for the specific rewrite formalism of *Higher-order Rewrite Systems (HRSs)*, following Saillard [40, Definition 5.2] we can encode the formalism of Dedukti as a specific HRS, allowing us to use their result in our setting. Alternatively, we refer to Férey's PhD thesis [26], which revisits classic confluence criteria in the rewrite formalism of Dedukti.

Confluence of $\beta\mathcal{R}_{\mathsf{cc}}$

Putting everything together, we obtain the confluence of $\beta\mathcal{R}_{\mathsf{cc}}$.

▶ **Theorem 14.** *$\beta\mathcal{R}_{\mathsf{cc}}$ is confluent.*

Proof. By Corollary 10 and Proposition 13 we have the confluence of $\beta\mathcal{R}_1$ and $\mathcal{R}_2$, and moreover the rewrite systems are left-linear and there are no critical pairs between them. Therefore, we conclude the confluence of their union by applying van Oostrom and van Raamsdonk's orthogonal combinations criterion [47, Theorem 3.13].[4] ◀

We obtain the following useful corollary, which we implicitly use in the rest of the article.

▶ **Corollary 15** (Injectivity of undefined symbols)**.** *If c is a constant that does not appear in the head of a rewrite rule, then $c\ t_1 ... t_k \equiv c\ u_1 ... u_k$ implies $t_i \equiv u_i$ for $i = 1..k$.*

5.2 Subject reduction

We start with subject reduction for β. Because we have already shown confluence of $\beta\mathcal{R}_{\mathsf{cc}}$, we obtain directly the injectivity of function types: if $(x : A) \to B \equiv (x : A') \to B'$ then $A \equiv A'$ and $B \equiv B'$. This is sufficient in order to ensure that β satisfies subject reduction.

▶ **Proposition 16** (SR_β)**.** *If $\Gamma \vdash t : A$ and $t \longrightarrow_\beta t'$ then $\Gamma \vdash t' : A$.*

Proof. Follows from the injectivity of function types [12, Lemma 31]. ◀

Moving to subject reduction for $\mathcal{R}_{\mathsf{cc}}$, the first point we realize is that this property does not hold unconditionally. For instance, the rule

$$\pi^{(\mathsf{S}\ \mathtt{l}_1)}_{0,\mathtt{l}_2}\ (\uparrow_{\_} \blacklozenge \mathsf{A})\ \mathsf{B} \longmapsto \pi^{\mathtt{l}_1}_{0,(\mathsf{S}\ \mathtt{l}_2)}\ \mathsf{A}\ \mathsf{B}$$

only preserves typing if $\mathsf{S}\ (\mathtt{l}_1[\theta] \vee (\mathtt{l}_1[\theta] + \mathtt{l}_2[\theta])) \equiv \mathtt{l}_1[\theta] \vee \mathsf{S}\ (\mathtt{l}_1[\theta] + \mathtt{l}_2[\theta])$, yet both sides are already in normal form. We could try to make the two sides convertible by adding a rewrite rule, however this rule would not be left-linear and thus make proving confluence much harder. Nevertheless, the fact that these terms are not convertible is actually not a problem because whenever $\mathtt{l}_1$ and $\mathtt{l}_2$ are substituted by terms of the form $\underline{n}$ for some $n \in \mathbb{N}$ then we see that the equation holds. Starting from this insight, we now show that subject reduction holds in a restricted form, which turns out to be sufficient for our needs.

We say that a term is *guarded* when all occurrences of $\Uparrow$ are of the form $\Uparrow_{\underline{n}}$ and all occurrences of π are of the form $\pi^{n_0}_{\underline{n_1},\underline{n_2}}$ for some $n, n_0, n_1, n_2 \in \mathbb{N}$. The set of guarded terms satisfies the following basic stability properties.

▶ **Proposition 17** (Stability of guarded terms under substitution and reduction)**.**

1. *If t, u are guarded then $t[u/x]$ is guarded.*
2. *If t is guarded and $t \longrightarrow t'$ then t' is guarded.*

We can now show that $\mathcal{R}_{\mathsf{cc}}$ satisfies subject reduction for guarded terms.

▶ **Proposition 18** ($\mathsf{SR}_{\mathcal{R}_{\mathsf{cc}}}$)**.** *If t is guarded and $\Gamma \vdash t : A$ and $t \longrightarrow_{\mathcal{R}_{\mathsf{cc}}} t'$ then $\Gamma \vdash t' : A$.*

[4] The same observation as in Footnote 3 applies here.

Proof. We use Lambdapi to automatically verify that the rules preserve typing (the correctness of this verification relies on the confluence of the rewrite system [41, 13], which we have by Theorem 14). The verification succeeds for all rules $l \longmapsto r \in \mathcal{R}_{\mathsf{cc}}$, except for those which do not preserve typing unconditionally. For these cases, Lambdapi reports conversion constraints on the substitution θ under which $\Gamma \vdash l[\theta] : A$ implies $\Gamma \vdash r[\theta] : A$.

1. Case $\Uparrow_{\mathrm{El}}$. Preserves typing if $\mathtt{l}_2[\theta] - \mathtt{l}_2[\theta] \equiv 0$. But by inversion of typing of the left-hand side we also get $\mathtt{l}_1[\theta] \equiv \mathtt{l}_2[\theta]$, so the rule preserves typing whenever $\mathtt{l}_1[\theta] - \mathtt{l}_1[\theta] \equiv 0$.
2. Case $\uparrow^2_\pi$. Preserves typing if $\mathrm{S}\ (\mathtt{l}_1[\theta] \vee (\mathtt{l}_1[\theta] + \mathtt{l}_2[\theta])) \equiv \mathtt{l}_1[\theta] \vee \mathrm{S}\ (\mathtt{l}_1[\theta] + \mathtt{l}_2[\theta])$.
3. Case $\uparrow^3_\pi$. Preserves typing if $(\mathtt{l}_1[\theta] + \mathtt{l}_2[\theta]) \vee \mathtt{l}_1[\theta] \equiv \mathtt{l}_1[\theta] + \mathtt{l}_2[\theta]$ and $\mathrm{S}\ (\mathtt{l}_1[\theta] + \mathtt{l}_2[\theta]) \vee \mathtt{l}_1[\theta] \equiv \mathrm{S}\ (\mathtt{l}_1[\theta] + \mathtt{l}_2[\theta])$.
4. Case $\uparrow^4_\pi$. Preserves typing if $\mathrm{S}\ (\mathtt{l}_1[\theta] + \mathtt{l}_2[\theta]) \vee \mathtt{l}_1[\theta] \equiv \mathrm{S}\ ((\mathtt{l}_1[\theta] + \mathtt{l}_2[\theta]) \vee \mathtt{l}_1[\theta])$.
5. Case $\uparrow^5_\pi$. Preserves typing if $\mathtt{l}_1[\theta] \vee \mathrm{S}\ (\mathtt{l}_1[\theta] + \mathtt{l}_2[\theta]) \equiv \mathrm{S}\ (\mathtt{l}_1[\theta] + \mathtt{l}_2[\theta])$ and $\mathfrak{R}\ \mathtt{l}_1[\theta]\ (\mathtt{l}_1[\theta] + \mathtt{l}_2[\theta]) \equiv \mathtt{l}_1[\theta] + \mathtt{l}_2[\theta]$.

Because t is guarded, it follows that $\mathtt{l}_1[\theta]$ is a concrete sort in case 1, and both $\mathtt{l}_1[\theta]$ and $\mathtt{l}_2[\theta]$ are concrete sorts in the other cases, so the result follows from the fact that these equations all hold for natural numbers. ◀

▶ **Corollary 19** ($\mathsf{SR}_{\beta\mathcal{R}_{\mathsf{cc}}}$). *If t is guarded and $\Gamma \vdash t : A$ and $t \longrightarrow^* t'$ then $\Gamma \vdash t' : A$.*

▶ Remark 20. Corollary 19 guarantees that the usual type inference algorithm for Dedukti [41] is sound when Γ and t are guarded. Indeed, by inspection on its definition, if the inputs Γ and t are guarded then only guarded terms are ever reduced.

6 The translation function

Defining a Dedukti encoding usually requires specifying a *translation function* from the syntax of the source system to the one of the framework. However, whereas cumulativity is *implicit* in CC, in Dedukti it is made *explicit* by the use of a lift ($\uparrow$). Therefore, when translating a CC term, the translation function needs to figure out when to insert such lifts, even though the initial term contains no information about cumulativity. To handle this, a first idea could be to define this function only for well-typed CC terms and use typing to retrieve the missing information. However, it is not clear how to define such a function in a unique and well-founded way – see Section 9 for a detailed discussion on why.

To solve this problem, we adapt the approach of Winterhalter et al. [48] of relying instead on an *inverse translation function* $|-|$, defined from a subset of the syntax of the framework to the syntax of CC. Because the syntax of Dedukti is more explicit than the one of CC, this function can be straightforwardly defined by structural induction. Then, we can use it to state and prove soundness and conservativity. Finally, the *direct* translation function can then be recovered as the underlying algorithm of our constructive proof of soundness.

We start by carving out a subset of Dedukti's syntax over which we define $|-|$. These are the *object terms* and *object contexts*, defined by the following grammars, and where n, m ranges over natural numbers and G ranges over arbitrary guarded terms.

$$\boxed{\Lambda_o} \ni \quad t, u, A, B ::= x \mid \lambda x : \mathrm{El}_{\underline{n}}\ A.t \mid \mathrm{u}_{\underline{n}} \mid \pi^{0}_{\underline{n},\underline{m}}\ A\ \lambda x : G.B \mid \uparrow_{\underline{n}}\ G\ t \mid t\ u$$

$$\boxed{\mathsf{Ctx}_o} \ni \quad \Gamma ::= \cdot \mid \Gamma, x : \mathrm{El}_{\underline{n}}\ A$$

The *inverse translation function* can then be defined by structural induction over object terms and contexts, by the following clauses.

$$
\begin{array}{ll}
|-| : \Lambda_o \to \Lambda_{\mathsf{cc}} & \|-\| : \mathsf{Ctx}_o \to \mathsf{Ctx}_{\mathsf{cc}} \\
|x| := x & \|\cdot\| := \cdot \\
|\mathrm{u}_{\underline{n}}| := n & \|\Gamma, x : \mathrm{El}_{\underline{n}}\ A\| := \|\Gamma\|, x : |A| \\
|\lambda x : \mathrm{El}_{\underline{n}}\ A.t| := \lambda x : |A|.|t| & \\
|\pi^{0}_{\underline{n},\underline{m}}\ A\ (\lambda x : G.B)| := \Pi x : |A|.|B| & \\
|\uparrow_{\underline{n}}\ G\ t| := |t| & \\
|t\ u| := |t|\ |u| & (t\ u \text{ not of previous forms})
\end{array}
$$

Crucially, object terms are all guarded, ensuring that whenever they are well typed then their reducts also are. In addition, object terms are stable under substitution, which moreover commutes with $|-|$, two basic properties that will be essential to our proofs.

▶ **Proposition 21** (Basic properties of Λ_o and $|-|$).
1. *If $t \in \Lambda_o$ then t is guarded.*
2. *If $t, u \in \Lambda_o$ then $t[u/x] \in \Lambda_o$ and $|t|[|u|/x] = |t[u/x]|$.*

7 Soundness

Our proof of soundness requires multiple intermediate steps. We start by showing the injectivity modulo lifting of El (Proposition 23) and the injectivity of $\Rightarrow$ (Proposition 24), two technical results that are then used in the proof of *coherence* (Theorem 26), ensuring that any two different Dedukti representations of the same CC term must be convertible. With coherence in hand, we can then show that the conversion relation of CC can be reflected by the inverse translation function into the framework (Proposition 28), which then finally allow us to show the soundness of our encoding (Theorem 31).

7.1 Injectivity

We start with the following generalization of Assaf's full reflection equations, used in the proof of the injectivity of El modulo lifting. From now on, let us write $(\uparrow_{\_}\ D)^k\ t$ for $\uparrow_{l_1}\ D\ (...(\uparrow_{l_k}\ D\ t)...)$ where the $l_1, ..., l_k$ can be any terms.

▶ **Lemma 22** (Generalized full reflection). *For all $k_1, k_2, n_1, n_2 \in \mathbb{N}$ we have*

$$\pi^{0}_{\underline{k_1+n_1},\underline{k_2+n_2}}\ ((\uparrow_{\_} \blacklozenge)^{k_1}\ A)\ (\lambda x : C.(\uparrow_{\_} \blacklozenge)^{k_2}\ B) \equiv \uparrow^{\frac{\mathfrak{R}(n_1+k_1,n_2+k_2)}{\mathfrak{R}(n_1,n_2)}} \blacklozenge\ (\pi^{0}_{\underline{n_1},\underline{n_2}}\ A\ (\lambda x : C.B))$$

Proof. By induction on $k_1 + k_2$, using Proposition 3. ◀

In the following, we use the greek letter ρ to refer to rewrite sequences $t \longrightarrow^* u$. Given a rewrite sequence ρ, we write $\hbar\rho$ for the first rewrite rule applied in the head in ρ or $\hbar\rho = \bot$ if no step takes place at the head, and we write $\#\rho$ for the total number of rewrite steps in ρ. For instance, if ρ denotes the sequence

$$\mathrm{El}_l\ ((\lambda x.\uparrow_{l'} \blacklozenge x)\ \mathrm{u}_{\underline{0}}) \longrightarrow \mathrm{El}_l\ (\uparrow_{l'} \blacklozenge \mathrm{u}_{\underline{0}}) \longrightarrow \mathrm{El}_{(\mathrm{P}\ l)}\ \mathrm{u}_{\underline{0}} \longrightarrow \mathrm{U}_{\underline{0}}$$

then we have $\#\rho = 3$ and $\hbar\rho = \uparrow_{\mathrm{El}}$, which is the rule applied in the middle.

We can now show that the constant El is injective modulo the insertion of some lifts.

▶ **Proposition 23** (Injectivity of El modulo lifting). *If* $\mathrm{El}_{l_1}\ A_1 \equiv \mathrm{El}_{l_2}\ A_2$*, where both sides are guarded and well typed, then there are natural numbers* k_1, k_2 *such that*
(1) $A_1 \equiv (\uparrow\_\ \blacklozenge)^{k_1}\ A_0$ *and* $A_2 \equiv (\uparrow\_\ \blacklozenge)^{k_2}\ A_0$ *for some term* A_0*.*
(2) $\mathrm{S}^{k_1}\ l_0 \equiv l_1$ *and* $\mathrm{S}^{k_2}\ l_0 \equiv l_2$ *for some term* l_0*.*

Proof. Note that, under the hypotheses of the lemma, (1) implies (2), so we proceed to show that the hypotheses imply (1), however when applying the i.h. we also obtain (2) for free.

By confluence we have $\mathrm{El}_{l_1}\ A_1 \longrightarrow^* B\ {}^*\!\longleftarrow \mathrm{El}_{l_2}\ A_2$ for some B. Writing ρ_1 for $\mathrm{El}_{l_1}\ A_1 \longrightarrow^* B$ and ρ_2 for $\mathrm{El}_{l_2}\ A_2 \longrightarrow^* B$, we show the result by induction on $\#\rho_1 + \#\rho_2$, and by case analysis on $\hbar\rho_1$ and $\hbar\rho_2$. If $\hbar\rho_1$ or $\hbar\rho_2$ is $\uparrow_{\mathrm{El}}$ or $\Uparrow_{\mathrm{El}}$ then the result is easily discharged using the induction hypothesis. Otherwise, we must have $\hbar\rho_1 = \hbar\rho_2$, and the only possibilities are $\bot$ or El_{u} or El_{π}. If $\hbar\rho_1 = \hbar\rho_2 = \bot$ then the result is easily shown, and if $\hbar\rho_1 = \hbar\rho_2 = \mathrm{El}_{\mathrm{u}}$ then the result follows by using the injectivity of U (which is an undefined constant). We now illustrate the more intricate case, when $\hbar\rho_1 = \hbar\rho_2 = \mathrm{El}_{\pi}$.

For $i = 1, 2$ we can decompose ρ_i as

$$\mathrm{El}_{l_i}\ A_i \longrightarrow^* \mathrm{El}_{l'_i}\ (\pi^{m_i}_{\underline{n_i^a},\underline{n_i^b}}\ A_i^a\ \lambda x : C_i.A_i^b) \longrightarrow (x : \mathrm{El}_{(\underline{m_i + n_i^a})}\ A_i^a) \to \mathrm{El}_{(\underline{m_i + n_i^b})}\ A_i^b \overset{\rho'_i}{\longrightarrow}{}^* B$$

where the first arguments of π must be concrete sorts because these are reducts of guarded terms. In the following, we write δ for either a or b. Then it must be the case that B is of the form $(x : B^a) \to B^b$ and that we can decompose ρ'_1 and ρ'_2 into $\rho_1^a, \rho_1^b, \rho_2^a, \rho_2^b$ given by

$$\mathrm{El}_{(\underline{m_1 + n_1^\delta})}\ A_1^\delta \overset{\rho_1^\delta}{\longrightarrow}{}^* B^\delta\ {}^*\overset{\rho_2^\delta}{\longleftarrow} \mathrm{El}_{(\underline{m_2 + n_2^\delta})}\ A_2^\delta$$

We have $\#\rho_1^\delta + \#\rho_2^\delta < \#\rho_1 + \#\rho_2$, therefore by i.h. we deduce that for some terms A_0^δ, l_0^δ and natural numbers k_1^δ, k_2^δ we have (a) $A_1^\delta \equiv (\uparrow\_\ \blacklozenge)^{k_1^\delta}\ A_0^\delta$ and $A_2^\delta \equiv (\uparrow\_\ \blacklozenge)^{k_2^\delta}\ A_0^\delta$, and moreover also (b) $\underline{m_1 + n_1^\delta} \equiv \mathrm{S}^{k_1^\delta}\ l_0^\delta$ and $\underline{m_2 + n_2^\delta} \equiv \mathrm{S}^{k_2^\delta}\ l_0^\delta$.

Because $\underline{m_1 + n_1^\delta} \longrightarrow^* \underline{m_1 + n_1^\delta}$, by confluence it follows that l_0^δ also reduces to a concrete sort $p^\delta \in \mathbb{N}$. We therefore have $m_1 + n_1^\delta = k_1^\delta + p^\delta$ and $m_2 + n_2^\delta = k_2^\delta + p^\delta$. Together with the equations from (a), this allows us to show the following for $i = 1, 2$.

$$\begin{aligned} A_i \equiv \pi^{m_i}_{\underline{n_i^a},\underline{n_i^b}}\ A_i^a\ \lambda x : C_i.A_i^b &\equiv \pi^{0}_{\underline{m_i + n_i^a},\underline{m_i + n_i^b}}\ A_i^a\ \lambda x : C_i.A_i^b \\ &\equiv \pi^{0}_{\underline{k_i^a + p^a},\underline{k_i^b + p^b}}\ ((\uparrow\_\ \blacklozenge)^{k_i^a}\ A_0^a)\ (\lambda x : C_i.(\uparrow\_\ \blacklozenge)^{k_i^b}\ A_0^b) \\ &\equiv \uparrow^{\mathfrak{R}(p^a + k_i^a, p^b + k_i^b)}_{\underline{\mathfrak{R}(p^a, p^b)}} \blacklozenge\ (\pi^{0}_{\underline{p^a},\underline{p^b}}\ A_0^a\ (\lambda x : C_i.A_0^b)) \end{aligned}$$

where the last equation follows from Lemma 22. It suffices now to show that $C_1 \equiv C_2$. To see this, note that by typing constraints we must have $C_i \equiv \mathrm{El}_{\underline{m_i + n_i^a}}\ A_i^a$ and thus

$$C_i \equiv \mathrm{El}_{\underline{k_i^a + p^a}}\ ((\uparrow\_\ \blacklozenge)^{k_i^a}\ A_0^a) \equiv \mathrm{El}_{\underline{p^a}} A_0^a$$

where the right-hand side does not depend on i. ◀

The injectivity of El modulo lifting is then used to establish the injectivity of $\Rightarrow$.

▶ **Proposition 24** (Injectivity of $\Rightarrow$). *If* $D_1 \Rightarrow l_1 \equiv D_2 \Rightarrow l_2$ *and both sides are well typed and guarded, then* $D_1 \equiv D_2$ *and* $l_1 \equiv l_2$*.*

Proof. The strategy is similar to the one employed in Proposition 23. By confluence we have $D_1 \Rightarrow l_1 \longrightarrow^* B\ {}^*\!\longleftarrow D_2 \Rightarrow l_2$ for some B. By writing ρ_i for $D_i \Rightarrow l_i \longrightarrow^* B$, we show the result by induction on $\#\rho_1$ and case analysis on $\hbar\rho_1$ (which must be the same as $\hbar\rho_2$). The case $\hbar\rho_1 = \bot$ is easy, and $\hbar\rho_1 = \Rightarrow_{\blacklozenge}$ follows by injectivity of U. Finally, the case $\Rightarrow_{\blacktriangleleft}$ follows by the induction hypotheses, typing constraints and Proposition 23. ◀

7.2 Coherence

To show coherence, we first need the following technical lemma, allowing to decompose a telescope D when $D \Rightarrow l$ is convertible to a function type.

▶ **Lemma 25** (Telescope decomposition). *If* $D \Rightarrow l \equiv (x : P) \to Q$ *then* $D \longrightarrow^* A\ {}_{l'} \blacktriangleleft \lambda x : C.D'$ *for some* A, l', C, D' *with* $P \equiv \mathrm{El}_{l'}\ A$ *and* $Q \equiv D' \Rightarrow l$.

Proof. By confluence, we have $D \Rightarrow l \longrightarrow^* B \ {}^*\!\longleftarrow (x : P) \to Q$. We must have B of the form $(x : P') \to Q'$ with $P' \equiv P$ and $Q' \equiv Q$, and we can decompose $D \Rightarrow l \longrightarrow^* B$ as

$$D \Rightarrow l \longrightarrow^* (A\ {}_{l'} \blacktriangleleft \lambda x : C.D') \Rightarrow l'' \longrightarrow (x : \mathrm{El}_{l'}\ A) \to D' \Rightarrow l'' \longrightarrow^* (x : P') \to Q'$$

We thus have $D \longrightarrow^* A\ {}_{l'} \blacktriangleleft \lambda x : C.D'$ and $\mathrm{El}_{l'}\ A \equiv P' \equiv P$ and $D' \Rightarrow l \equiv Q' \equiv Q$. ◀

We now move to the proof of *coherence*, the central auxiliary result needed for soundness, ensuring that any two different Dedukti representations of the same CC term must be convertible. The actual statement of the theorem is however a bit more intricate.

▶ **Theorem 26** (Coherence). *Let* $t_1, t_2 \in \Lambda_o$ *with* $\Gamma \vdash t_1 : A_1$ *and* $\Gamma \vdash t_2 : A_2$. *If* $|t_1| = |t_2|$ *then at least one of the following holds:*

(1) $t_1 \equiv t_2$
(2) $\Gamma \vdash \uparrow^{\underline{m}}_{\underline{n}}\ D\ t_2 : D \Rightarrow \underline{m}$ *and* $t_1 \equiv \uparrow^{\underline{m}}_{\underline{n}}\ D\ t_2$ *for some* D *guarded*
(3) $\Gamma \vdash \uparrow^{\underline{m}}_{\underline{n}}\ D\ t_1 : D \Rightarrow \underline{m}$ *and* $t_2 \equiv \uparrow^{\underline{m}}_{\underline{n}}\ D\ t_1$ *for some* D *guarded*

Proof. The proof is by induction on t_1 and t_2, following the definition of $|-|$.

- Case $t_1 = \uparrow_{\underline{n}}\ D\ u$. By inversion of typing, uniqueness of type and injectivity of function types, we have $\Gamma \vdash D : \mathrm{Tele}$ and $\Gamma \vdash u : D \Rightarrow \underline{n}$. By i.h. on u and t_2, we have three cases to consider.
  - **(a)** $u \equiv t_2$. By confluence, u and t_2 have a common reduct w. Using subject reduction we know w has both types $D \Rightarrow \underline{n}$ and A_2 so by uniqueness of type, we know $D \Rightarrow \underline{n} \equiv A_2$ so we can conclude that $\Gamma \vdash t_2 : D \Rightarrow \underline{n}$ and thus that $\Gamma \vdash \uparrow_{\underline{n}}\ D\ t_2 : D \Rightarrow (\mathrm{S}\ \underline{n})$. Knowing that $t_1 \equiv \uparrow_{\underline{n}}\ D\ t_2$ by congruence, we conclude.
  - **(b)** $\Gamma \vdash \uparrow^{\underline{m'}}_{\underline{n'}}\ D'\ t_2 : D' \Rightarrow \underline{m'}$ and $u \equiv \uparrow^{\underline{m'}}_{\underline{n'}}\ D'\ t_2$. Similarly to above, we can show $D \Rightarrow \underline{n} \equiv D' \Rightarrow \underline{m'}$ by confluence, subject reduction and uniqueness of type. By injectivity of $\Rightarrow$ (Proposition 24) we get $D \equiv D'$ and $\underline{n} \equiv \underline{m'}$ which means $n = m'$ given that they are concrete. So $t_1 = \uparrow_{\underline{n}}\ D\ u \equiv \uparrow_{\underline{n}}\ D\ (\uparrow^{\underline{n}}_{\underline{n'}}\ D\ t_2) = \uparrow^{\underline{1+n}}_{\underline{n'}}\ D\ t_2$ by folding notations. Finally, we have $\Gamma \vdash t_2 : D' \Rightarrow \underline{n'}$, so by conversion we get $\Gamma \vdash t_2 : D \Rightarrow \underline{n'}$ and thus $\Gamma \vdash \uparrow^{\underline{1+n}}_{\underline{n'}}\ D\ t_2 : D \Rightarrow \underline{1+n}$.
  - **(c)** $\Gamma \vdash \uparrow^{\underline{m'}}_{\underline{n'}}\ D'\ u : D' \Rightarrow \underline{m'}$ and $t_2 \equiv \uparrow^{\underline{m'}}_{\underline{n'}}\ D'\ u$. This gives us in particular that $\Gamma \vdash u : D' \Rightarrow \underline{n'}$ so by uniqueness of type we get $D \Rightarrow \underline{n} \equiv D' \Rightarrow \underline{n'}$ and thus $D \equiv D'$ and $n = n'$. If $m' = n$ then we have $t_2 \equiv u$ so we proceed as in case (a), otherwise $m' \geqslant 1 + n$ so we can conclude with $t_2 \equiv \uparrow^{\underline{m'}}_{\underline{n}}\ D\ u = \uparrow^{\underline{m'}}_{\underline{1+n}}\ D\ (\uparrow_{\underline{n}}\ D\ u) = \uparrow^{\underline{m'}}_{\underline{1+n}}\ D\ t_1$ and $\Gamma \vdash \uparrow^{\underline{m'}}_{\underline{1+n}}\ D\ t_1 : D \Rightarrow \underline{m'}$.

The case $t_2 = \uparrow_{\underline{n}}\ D\ u$ follows by the same reasoning, and for the other cases the definition of $|-|$ imposes that t_1 and t_2 must have the same head structure. Therefore, to conclude we consider t_1 and t_2 of the same form. We illustrate the following case:

- Case $t_1 = u_1\ v_1$ and $t_2 = u_2\ v_2$. By inversion we have $\Gamma \vdash u_i : (x : A_i) \to B_i$ and $\Gamma \vdash v_i : A_i$. By the i.h. applied to u_1 and u_2, we have three cases to consider:

(a) $u_1 \equiv u_2$. We thus get $A_1 \equiv A_2$ and $B_1 \equiv B_2$. Looking at the induction hypothesis on v_1 and v_2, in all cases we must have $v_1 \equiv v_2$. Indeed, if we are in cases (2) or (3) then we get $A_1 \equiv D \Rightarrow \underline{p}$ and $A_2 \equiv D \Rightarrow \underline{q}$, but together with $A_1 \equiv A_2$ this implies $p = q$, meaning that no lifts are inserted between v_1 and v_2. We thus conclude that $t_1 \equiv t_2$.

(b) $\Gamma \vdash \uparrow^m_{\underline{n}} D\ u_2 : D \Rightarrow \underline{m}$ and $u_1 \equiv \uparrow^m_{\underline{n}} D\ u_2$. Now, $(x : A_1) \to B_1 \equiv D \Rightarrow \underline{m}$, so by Lemma 25 we have $D \longrightarrow^* a\ _l \blacktriangleleft \lambda x : C.D'$ with $\mathrm{El}_l\ a \equiv A_1$ and $B_1 \equiv D' \Rightarrow \underline{m}$. Moreover, we also get that $\mathrm{El}_l\ a \equiv A_2$ and $B_2 \equiv D' \Rightarrow \underline{n}$. We are again in a situation where v_1 and v_2 share a type, so by the same arguments as in case (a) the i.h. gives $v_1 \equiv v_2$. Therefore,

$$t_1 = u_1\ v_1 \equiv (\uparrow^m_{\underline{n}} (a\ _l \blacktriangleleft \lambda x : C.D')\ u_2)\ v_2 \equiv \uparrow^m_{\underline{n}} D'[v_2/x]\ (u_2\ v_2) = \uparrow^m_{\underline{n}} D'[v_2/x]\ t_2$$

For typing, we have $\Gamma \vdash t_2 : B_2[v_2/x]$ so by conversion we have $\Gamma \vdash t_2 : D'[v_2/x] \Rightarrow \underline{n}$ and thus $\Gamma \vdash \uparrow^m_{\underline{n}} D'[v_2/x]\ t_2 : D'[v_2/x] \Rightarrow \underline{m}$.

(c) $\Gamma \vdash \uparrow^m_{\underline{n}} D\ u_1 : D \Rightarrow \underline{m}$ and $u_2 \equiv \uparrow^m_{\underline{n}} D\ u_1$. Symmetric to case (b). ◀

7.3 Reflection of conversion

With coherence in hand, we can show that the conversion relation of CC can be reflected by the inverse translation function into the framework. As an intermediate lemma, we first need to show that individual reduction steps of CC can be simulated in Dedukti.

▶ **Lemma 27** (Simulation of reduction steps). *Let $t \in \Lambda_o$ with $\Gamma \vdash t : A$ and $|t| \longrightarrow u$ for some $u \in \Lambda_{\mathsf{cc}}$. Then, there is some $t' \in \Lambda_o$ such that $|t'| = u$ and $t \longrightarrow^* t'$.*

Proof. By induction on t, following the definition of Λ_o. Almost all cases are either impossible, or follow by applying the i.h. to the subterm being reduced. The only interesting case is when $t = t_1\ t_2$ and the reduction happens in the head. Then, the only possibility is that $t_1 = \uparrow_{\underline{n_k}} D_k\ (...(\uparrow_{\underline{n_1}} D_1\ v)...)$ with $v = \lambda x : C.s$ and $|t| = (\lambda x : |C|.|s|)\ |t_2| \longrightarrow |s|[|t_2|/x]$. If $k = 0$ then the result is immediate, as t is a β redex. Otherwise, by typing constraints and Proposition 24 we can see that we have $D_1 \equiv ... \equiv D_k$ and $n_{i+1} = n_i + 1$ for $i = 1..k-1$, so by confluence we have some common reduct D_0 of all of them so that $t_1 \longrightarrow^* \uparrow^{n_k+1}_{\underline{n_1}} D_0\ v$. Then, by inversion of typing, v has both types $D_0 \Rightarrow \underline{n_1}$ and $(x : C) \to A'$ for some A', hence by uniqueness of types we have $D_0 \Rightarrow \underline{n_1} \equiv (x : C) \to A'$, which by Lemma 25 implies $D_0 \longrightarrow^* C'\ _l \blacktriangleleft \lambda x : B.D'$ for some C', l, B, D'. Abbreviating $C'\ _l \blacktriangleleft \lambda x : B.D'$ as D'_0,

$$t \longrightarrow^* \uparrow^{n_k+1}_{\underline{n_1}} D'_0\ (\lambda x : C.s)\ t_2 \longrightarrow^* (\lambda x : C.\uparrow^{n_k+1}_{\underline{n_1}} D'\ s)\ t_2 \longrightarrow \uparrow^{n_k+1}_{\underline{n_1}} D'[t_2/x]\ s[t_2/x]$$

and we have $\uparrow^{n_k+1}_{\underline{n_1}} D'[t_2/x]\ s[t_2/x] \in \Lambda_o$, with $|\uparrow^{n_k+1}_{\underline{n_1}} D'[t_2/x]\ s[t_2/x]| = |s|[|t_2|/x]$. ◀

▶ **Proposition 28** (Reflection of type conversion). *Let $A, B \in \Lambda_o$ with $\Gamma \vdash A : \mathrm{U}_{\underline{n}}$ and $\Gamma \vdash B : \mathrm{U}_{\underline{m}}$. If $|A| \equiv |B|$ then $\mathrm{El}_{\underline{n}}\ A \equiv \mathrm{El}_{\underline{m}}\ B$.*

Proof. Take $k := \max\{n, m\}$; we have $\Gamma \vdash \uparrow^k_{\underline{n}} \blacklozenge A : \mathrm{U}_{\underline{k}}$ and $\Gamma \vdash \uparrow^k_{\underline{m}} \blacklozenge B : \mathrm{U}_{\underline{k}}$ and $|\uparrow^k_{\underline{n}} \blacklozenge A| = |A| \equiv |B| = |\uparrow^k_{\underline{m}} \blacklozenge B|$. By confluence we have $|\uparrow^k_{\underline{n}} \blacklozenge A| \longrightarrow^* C\ {}^*\!\longleftarrow |\uparrow^k_{\underline{m}} \blacklozenge B|$ for some C. By iterating Lemma 27 with subject reduction, we get $\uparrow^k_{\underline{n}} \blacklozenge A \longrightarrow^* A'$ and $\uparrow^k_{\underline{m}} \blacklozenge B \longrightarrow^* B'$ and $|A'| = C = |B'|$ for some A' and B'. We also have $\Gamma \vdash A' : \mathrm{U}_{\underline{k}}$ and $\Gamma \vdash B' : \mathrm{U}_{\underline{k}}$, so by Theorem 26 we get $A' \equiv B'$ – note that because A' and B' have the same type, there can be no lifts between them. Therefore, we have $\uparrow^k_{\underline{n}} \blacklozenge A \equiv \uparrow^k_{\underline{m}} \blacklozenge B$ and thus we conclude $\mathrm{El}_{\underline{n}}\ A \equiv \mathrm{El}_{\underline{k}}\ (\uparrow^k_{\underline{n}} \blacklozenge A) \equiv \mathrm{El}_{\underline{k}}\ (\uparrow^k_{\underline{m}} \blacklozenge B) \equiv \mathrm{El}_{\underline{m}}\ B$. ◀

7.4 Soundness

We now have almost all auxiliary results needed for showing soundness. As a last step, we only need the following two easy lemmas.

▶ **Lemma 29** (Computing the El of a translation). *Let $A \in \Lambda_o$ with $\mathrm{El}_l\ A$ well typed.*
1. *If $|A| = n$ then $\mathrm{El}_l\ A \longrightarrow^* \mathrm{U}_{\underline{n}}$.*
2. *If $|A| = \Pi x : A_1.A_2$ then $\mathrm{El}_l\ A \longrightarrow^* (x : \mathrm{El}_{\underline{n_1}}\ A_1') \to \mathrm{El}_{\underline{n_2}}\ A_2'$ with $|A_i'| = A_i$.*

Proof. By definition of $|-|$ and typing constraints. ◀

▶ **Lemma 30** (Telescope translation). *Let $A_1, A_2 \in \Lambda_o$ with $\Gamma \vdash A_i : \mathrm{U}_{\underline{n_i}}$. If $|A_i| = \Delta \Rightarrow m_i$ for some $m_1 \leqslant m_2$, then we have $\mathrm{El}_{\underline{n_i}}\ A_i \equiv D \Rightarrow \underline{m_i}$ for some guarded D with $\Gamma \vdash D : \mathrm{Tele}$.*

Proof. By induction on Δ. ◀

▶ **Theorem 31** (Soundness). *If $\Gamma \vdash_{\mathsf{cc}} t : A$ then we have $\Gamma' \vdash t' : \mathrm{El}_{\underline{n}}\ A'$ for some $\Gamma' \in \mathsf{Ctx}_o$ and $t', A' \in \Lambda_o$ and $n \in \mathbb{N}$ with $\|\Gamma'\| = \Gamma$ and $|t'| = t$ and $|A'| = A$.*

Proof. We instead show the following two points, which together imply the theorem.
- If $\Gamma \vdash_{\mathsf{cc}}$ then $\Gamma' \vdash$ for some $\Gamma' \in \mathsf{Ctx}_o$ with $\|\Gamma'\| = \Gamma$.
- If $\Gamma \vdash_{\mathsf{cc}} t : A$ and $\Gamma' \vdash$ for some $\Gamma' \in \mathsf{Ctx}_o$ with $\|\Gamma'\| = \Gamma$ then $\Gamma' \vdash t' : \mathrm{El}_{\underline{n}}\ A'$ for some $n \in \mathbb{N}$ and $A', t' \in \Lambda_o$ with $|A'| = A$ and $|t'| = t$.

We prove them by induction on the derivation of $\Gamma \vdash_{\mathsf{cc}}$ or $\Gamma \vdash_{\mathsf{cc}} t : A$, and illustrate here two interesting cases.

- Case

$$\textsc{Lam}\ \frac{\Gamma \vdash_{\mathsf{cc}} A : n \qquad \Gamma, x : A \vdash_{\mathsf{cc}} t : B}{\Gamma \vdash_{\mathsf{cc}} \lambda x : A.t : \Pi x : A.B}$$

  By i.h. and Lemma 29 we have $\Gamma' \vdash A' : \mathrm{U}_{\underline{n}}$ and $|A'| = A$. Therefore we have $\Gamma', x : \mathrm{El}_{\underline{n}}\ A' \vdash$, so by i.h. we get $\Gamma', x : \mathrm{El}_{\underline{n}}\ A' \vdash t' : \mathrm{El}_{\underline{m}}\ B'$ for some m and with $|t'| = t$ and $|B'| = B$. By inversion, we then deduce $\Gamma', x : \mathrm{El}_{\underline{n}}\ A' \vdash B' : \mathrm{U}_{\underline{m}}$. We can now show $\Gamma' \vdash \lambda x : \mathrm{El}_{\underline{n}}\ A'.t' : (x : \mathrm{El}_{\underline{n}}\ A') \to \mathrm{El}_{\underline{m}}\ B'$ and because its type is convertible to $\mathrm{El}_{\underline{\Re(n,m)}}\ (\pi^{0}_{\underline{n},\underline{m}}\ A'\ (\lambda x : \mathrm{El}_{\underline{n}}\ A'.B'))$, which is well typed, we conclude by applying the conversion rule.
- Case

$$A \subseteq B\ \textsc{Conv}\ \frac{\Gamma \vdash_{\mathsf{cc}} t : A \qquad \Gamma \vdash_{\mathsf{cc}} B : n}{\Gamma \vdash_{\mathsf{cc}} t : B}$$

  By induction hypothesis we have $\Gamma' \vdash t' : \mathrm{El}_{\underline{m}}\ A'$ and $\Gamma' \vdash B' : \mathrm{U}_{\underline{n}}$ with $|t'| = t$, $|A'| = A$ and $|B'| = B$ (using Lemma 29 for the second derivation). By inversion we obtain $\Gamma' \vdash A' : \mathrm{U}_{\underline{m}}$. We now use Lemma 2 to split $A \subseteq B$ into two cases:
  - $A \equiv B$. We have $|A'| \equiv |B'|$ so by Proposition 28 we conclude $\mathrm{El}_{\underline{m}}\ A' \equiv \mathrm{El}_{\underline{n}}\ B'$, and thus $\Gamma' \vdash t' : \mathrm{El}_{\underline{n}}\ B'$.
  - $A \longrightarrow^* \Delta \Rightarrow p$ and $B \longrightarrow^* \Delta \Rightarrow q$ with $p \leqslant q$. We apply Lemma 27 on A' to get some A'' such that $|A''| = \Delta \Rightarrow p$ and $A' \longrightarrow^* A''$. Similarly, we get B'' with $|B''| = \Delta \Rightarrow q$ and $B' \longrightarrow^* B''$. We can then apply Lemma 30 to obtain a guarded term D such that $\Gamma' \vdash D : \mathrm{Tele}$ and $\mathrm{El}_{\underline{m}}\ A'' \equiv D \Rightarrow \underline{p}$ and $\mathrm{El}_{\underline{n}}\ B'' \equiv D \Rightarrow \underline{q}$. We can now conclude with $\Gamma' \vdash \uparrow^{\underline{q}}_{\underline{p}}\ D\ t : \mathrm{El}_{\underline{n}}\ B'$. ◀

8 Conservativity

Now that we have seen that our encoding is sound, we can move to the proof of conservativity. The usual statement of conservativity (using direct translation functions $[-] : \Lambda_{\mathsf{cc}} \to \Lambda_{\mathsf{dk}}$ and $[\![-]\!] : \mathsf{Ctx}_{\mathsf{cc}} \to \mathsf{Ctx}_{\mathsf{dk}}$) would say that, given Γ, A satisfying $\Gamma \vdash_{\mathsf{cc}} A : n$, if $[\![\Gamma]\!] \vdash t : \mathrm{El}_{\underline{n}}\ [A]$ then we have $\Gamma \vdash_{\mathsf{cc}} t' : A$ for some t'. When rephrasing this statement with the inverse translation function $|-|$, the full conservativity property would then assert that, for $\Gamma \in \mathsf{Ctx}_o$ and $A \in \Lambda_o$ with $\|\Gamma\| \vdash |A| : n$, if $\Gamma \vdash t : \mathrm{El}_{\underline{n}}\ A$ then $\|\Gamma\| \vdash_{\mathsf{cc}} t' : |A|$ for some t'.

In the following, we instead show *conservativity for object terms*, a restricted form of conservativity in which the witness t of the typing judgment $\Gamma \vdash t : \mathrm{El}_{\underline{n}}\ A$ is required to be an object term. We argue that this is enough because in practice the object terms are the only ones a user of the encoding (or an automatic translator) would write. Nevertheless, it should be possible to strengthen our result to obtain full conservativity, as discussed in the conclusion.

The first step in our proof is showing that $|-|$ preserves definitional equality. This is however not immediate, because $|-|$ does not preserve reduction steps. Fortunately, we can define an auxiliary function $|-|^\bullet$ extending $|-|$ that satisfies this property. We start by defining the *extended object terms* $\Lambda_o^\bullet$ which will be used as the domain of $|-|^\bullet$. Here we write G, G' for any guarded terms, and n, n_0, n_1, n_2 for any natural numbers.

$$\boxed{\Lambda_o^\bullet} \ni \quad t, u, A, B ::= x \mid (x : A) \to B \mid \lambda x : A.t \mid \mathrm{U}_{\underline{n}} \mid \mathrm{El}_G\ A \mid \mathrm{u}_{\underline{n}}$$
$$\mid \pi^{n_0}_{\underline{n_1},\underline{n_2}}\ A\ \lambda x : G.B \mid \uparrow_G\ G'\ t \mid \Uparrow_{\underline{n}}\ t \mid t\ u$$

The function $|-|^\bullet$ is then defined by the following clauses.

$$|-|^\bullet : \Lambda_o^\bullet \to \Lambda_{\mathsf{cc}}$$

$\lvert(x : A) \to B\rvert^\bullet := \Pi x : \lvert A\rvert^\bullet.\lvert B\rvert^\bullet$		
$\lvert x\rvert^\bullet := x$	$\lvert\mathrm{El}_G\ A\rvert^\bullet := \lvert A\rvert^\bullet$	$\lvert\lambda x : A.t\rvert^\bullet := \lambda x : \lvert A\rvert^\bullet.\lvert t\rvert^\bullet$
$\lvert\mathrm{u}_{\underline{n}}\rvert^\bullet := n$	$\lvert\uparrow_G\ G'\ t\rvert^\bullet := \lvert t\rvert^\bullet$	$\lvert\pi^{n_0}_{\underline{n_1},\underline{n_2}}\ A\ (\lambda x : G.B)\rvert^\bullet := \Pi x : \lvert A\rvert^\bullet.\lvert B\rvert^\bullet$
$\lvert\mathrm{U}_{\underline{n}}\rvert^\bullet := n$	$\lvert\Uparrow_{\underline{n}}\ t\rvert^\bullet := \lvert t\rvert^\bullet$	$\lvert t\ u\rvert^\bullet := \lvert t\rvert^\bullet\ \lvert u\rvert^\bullet$ ($t\ u$ not of previous forms)

We can show that $|-|^\bullet$ satisfies many desirable properties, among them being the preservation of reduction steps and thus also of definitional equality by $|-|^\bullet$.

▶ **Lemma 32** (Basic properties of $\Lambda_o^\bullet$ and $|-|^\bullet$).

1. *$\Lambda_o^\bullet$ is a superset of Λ_o, and $|-|^\bullet$ restricts to $|-|$ in Λ_o.*
2. *If $t \in \Lambda_o^\bullet$ then t is guarded.*
3. *If $t, u \in \Lambda_o^\bullet$ then $t[u/x] \in \Lambda_o^\bullet$ and $|t|^\bullet[|u|^\bullet/x] = |t[u/x]|^\bullet$.*
4. *If $t \in \Lambda_o^\bullet$ and $t \longrightarrow^* u$ then $u \in \Lambda_o^\bullet$ and $|t|^\bullet \longrightarrow^* |u|^\bullet$.*
5. *If $t, u \in \Lambda_o^\bullet$ and $t \equiv u$ then $|t|^\bullet \equiv |u|^\bullet$.*

Using these basic properties, we can now show conservativity.

▶ **Theorem 33** (Conservativity for object terms). *Let $\Gamma \in \mathsf{Ctx}_o$ and $A \in \Lambda_o$ with $\|\Gamma\| \vdash_{\mathsf{cc}} |A| : n$ for some n. If $\Gamma \vdash t : \mathrm{El}_{\underline{n}}\ A$ with t an object term, then we have $\|\Gamma\| \vdash_{\mathsf{cc}} |t| : |A|$.*

Proof. We instead show the following claim.

▷ Claim 34. Let $\Gamma \vdash t : A$ with $\Gamma \in \mathsf{Ctx}_o$ and $\|\Gamma\| \vdash_{\mathsf{cc}}$. If t is an object term, then there exists $A' \in \Lambda_o^\bullet$ with $A \equiv A'$ and $\|\Gamma\| \vdash_{\mathsf{cc}} |t| : |A'|^\bullet$.

First note that this implies the statement of the theorem. Indeed, by the claim we have $\|\Gamma\| \vdash_{\mathsf{cc}} |t| : |B|^\bullet$ for some $B \in \Lambda_o^\bullet$ with $B \equiv \mathrm{El}_{\underline{n}}\ A$. Therefore $|B|^\bullet \equiv |A|^\bullet = |A|$, so we conclude $\|\Gamma\| \vdash_{\mathsf{cc}} |t| : |A|$ by the conversion rule.

We proceed with the proof of the claim, by induction on t, following the definition of Λ_o. We illustrate the interesting case of λ-abstraction: $t = \lambda x : \mathrm{El}_{\underline{n}}\ A_1.u$. By inversion we have $\Gamma \vdash A_1 : \mathrm{U}_{\underline{n}}$ and $\Gamma, x : \mathrm{El}_{\underline{n}}\ A_1 \vdash u : A_2$ for some A_2 with $A \equiv (x : \mathrm{El}_{\underline{n}}\ A_1) \to A_2$. By i.h. we thus have $\|\Gamma\| \vdash_{\mathsf{cc}} |A_1| : |B_1|^\bullet$ with $B_1 \equiv \mathrm{U}_{\underline{n}}$. Therefore, we have $|B_1|^\bullet \equiv n$, so by conversion we can derive $\|\Gamma\| \vdash_{\mathsf{cc}} |A_1| : n$, and so $\|\Gamma\|, x : |A_1| \vdash_{\mathsf{cc}}$. By i.h. once more, we have $\|\Gamma\|, x : |A_1| \vdash_{\mathsf{cc}} |u| : |B_2|^\bullet$ for some B_2 with $B_2 \equiv A_2$. We can thus derive $\|\Gamma\| \vdash_{\mathsf{cc}} \lambda x : |A_1|.|u| : |(x : \mathrm{El}_{\underline{n}}\ A_1) \to B_2|^\bullet$ and $A \equiv (x : \mathrm{El}_{\underline{n}}\ A_1) \to B_2$. ◀

9 Related work

The first attempt to encode CC in Dedukti dates back to the work of Assaf. He first identified the full-reflection equations (discussed in Section 4) in earlier work studying a variant of the calculus of constructions with explicit cumulativity [4]. There, cumulativity is made explicit by a family of lifts $\uparrow_i : \mathsf{U}_i \to \mathsf{U}_{i+1}$, which are sufficient in his setting because the theory considered lacks product covariance.

These ideas were then employed in encoding a class of cumulative type systems (CTSs) in Dedukti [5], containing in particular the type system CC. In order to handle product covariance, he proposed the use of η-expansion at translation time: for instance, a variable $f : \mathsf{Nat} \to 0$ would be translated *at type* $\mathsf{Nat} \to 1$ as $\lambda x.\uparrow_{\underline{0}}\ (f\ x)$. This however turned out to invalidate conservativity, as observed by Thiré [45, Example 6.6].

Moreover, as mentioned in the introduction, the translation functions used by Assaf for stating and proving soundness turn out to be ill-defined. He mutually defines functions $[-]_\Gamma$ and $[-]_{\Gamma \vdash C}$ and $[\![-]\!]$, and among their defining clauses he states $[t]_{\Gamma \vdash C} := \lambda x : [\![A]\!].[t\ x]_{\Gamma, x:A \vdash B}$ if $C \equiv \Pi x : A.B$ and t has a principal type convertible to $\Pi x : A.B'$ with $B' \subsetneq B$. However, the term A is only determined up to conversion, yet the function is defined over unquontiented terms, and the preservation of conversion is only shown at a later stage. Worse, because A is recovered using typing information, it might not be structurally smaller than t, and no well-founded order is given to justify the recursive call of $[\![-]\!]$ on A.

Regarding confluence, Assaf actually relies in his presentation on an axiomatization of the conversion relation required for the encoding. Because in Dedukti the conversion must be implemented by rewrite rules, each instantiation of his encoding then also needs to provide a rewrite system correctly implementing these equational axioms. In the particular case of CC, Assaf provides rules for implementing them, yet they are not confluent since some critical pairs are not joinable. This problem was later fixed in his joint work with Dowek, Jouannaud and Liu [8], though it required the use of rewriting modulo ACU, which is less efficient and harder to implement than pure syntactic matching. The problems with soundness and conservativity remained unaddressed.

Some years after the work of Assaf, the problem regained attention and new encodings were proposed by Thiré [45], also supporting a class of CTSs, and Férey [26], also supporting universe polymorphism. Starting from Thiré's observation that η-expanding at translation time breaks conservativity, they decided to instead rely on a generalized cast operator mapping a term $t : \mathrm{El}_{l_a}\ a$ to ${}^{l_b}_{l_a}\uparrow^b_a\ e\ t : \mathrm{El}_{l_b}\ b$, where e is a term witnessing the inclusion of a in b. Unfortunately, the use of a multi-step lift then required non-left-linear rules to ensure that two consecutive casts can be composed or that identity casts can be removed. Despite the impressive work of Férey on confluence criteria for non-left-linear systems [25], they were unable to show the confluence of their encodings.

The translation function employed by Thiré unfortunately inherited the issue of Assaf's function, as it also makes recursive calls on terms obtained through typing information without giving a decreasing measure. The proposal of Férey uses however a different technique, and instead defines the translation function over typing derivations. However, because a typing judgment can be derived in multiple ways, he then needs to show that they are nevertheless translated to convertible terms in Dedukti, which corresponds in our case to the coherence property. To show this, he crucially relies on the way terms are represented in his encoding, as untyped codes annotated with a type. Because of this, it is not clear to us how his technique could be adapted to our case, which is why we chose to prove soundness using an inverse translation function, instead of defining a translation function over derivations. Finally, conservativity is stated only as a conjecture for both of Thiré's and Férey's encodings.

10 Conclusion

In this work we have given an encoding of CC in Dedukti satisfying the necessary properties for being used in practice, solving a longstanding open problem. Our proof of confluence combines many confluence criteria and heavily uses the automated tools developed by the community. Yet, at the present moment, none of the available tools are able to fully show our result by themselves. Proving the confluence of our system automatically can thus be an interesting challenge for the next generation of today's confluence checkers. A natural direction could be trying to automate Dowek et al.'s criterion [21], which is the only one we are aware of that can show the confluence of $\beta\mathcal{R}_1$ directly.

Our work has also identified a problem with the definition of the translation function in some previous attempts at encoding CC in Dedukti. To solve this issue, we have then contributed an adaptation of the technique of Winterhalter et al. [48] in which soundness is instead stated and proved using an inverse translation function.

Regarding conservativity, we have proven a restricted form concerning only object terms. Even though we believe that for practical needs our result is sufficient, we conjecture that full conservativity can be obtained by adapting the logical relations technique of Assaf [6]. Alternatively, we could modify our encoding and employ the technique described by Felicissimo [22], which allows for easy conservativity proofs at the cost of increasing the amount of type annotations in the syntax. There is already ongoing work on removing these annotations by incorporating bidirectional typing into Dedukti [23], yet the encoding presented here would not be covered by the presently available framework.

Finally, we believe that our work can be a starting point for incorporating Coq's universe-polymorphism. Among previous work, only Férey considers the combination of CC with universe polymorphism. Combining his ideas with ours is a promising direction to explore.

References

1 AProVE. URL: `https://aprove.informatik.rwth-aachen.de/`.

2 CeTA. URL: `http://cl-informatik.uibk.ac.at/software/ceta/`.

3 CSIho. URL: `http://cl-informatik.uibk.ac.at/software/csi/ho/`.

4 Ali Assaf. A calculus of constructions with explicit subtyping. In *20th International Conference on Types for Proofs and Programs (TYPES 2014)*, volume 39, 2014.

5 Ali Assaf. *A framework for defining computational higher-order logics*. These, École polytechnique, September 2015. URL: `https://pastel.archives-ouvertes.fr/tel-01235303`.

6 Ali Assaf. Conservativity of Embeddings in the lambda Pi Calculus Modulo Rewriting. In Thorsten Altenkirch, editor, *13th International Conference on Typed Lambda Calculi and Applications (TLCA 2015)*, volume 38 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 31–44, Dagstuhl, Germany, 2015. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.TLCA.2015.31`.

7 Ali Assaf, Guillaume Burel, Raphaël Cauderlier, David Delahaye, Gilles Dowek, Catherine Dubois, Frédéric Gilbert, Pierre Halmagrand, Olivier Hermant, and Ronan Saillard. Dedukti: a logical framework based on the λΠ-calculus modulo theory. Unpublished, 2016.

8 Ali Assaf, Gilles Dowek, Jean-Pierre Jouannaud, and Jiaxiang Liu. Untyped Confluence In Dependent Type Theories. working paper or preprint, April 2017. URL: `https://hal.inria.fr/hal-01515505`.

9 H.P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. Studies in Logic and the Foundations of Mathematics. Elsevier Science, 1984.

10 Bruno Barras. Auto-validation d'un systeme de preuves avec familles inductives. *These de doctorat, Université Paris*, 7, 1999.

11 M. Bezem, J.W. Klop, R. de Vrijer, and Terese. *Term Rewriting Systems*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2003. URL: `https://books.google.fr/books?id=7QQ5u-4tRUkC`.

12 Frédéric Blanqui. *Théorie des types et réécriture. (Type theory and rewriting)*. PhD thesis, University of Paris-Sud, Orsay, France, 2001. URL: `https://tel.archives-ouvertes.fr/tel-00105522`.

13 Frédéric Blanqui. Type safety of rewrite rules in dependent types. In *5th International Conference on Formal Structures for Computation and Deduction*, 2020.

14 Frédéric Blanqui. Encoding type universes without using matching modulo AC. In *Proceedings of the 7th International Conference on Formal Structures for Computation and Deduction*, Leibniz International Proceedings in Informatics 228, 2022. `doi:10.4230/LIPIcs.FSCD.2022.24`.

15 Frédéric Blanqui. HOL-Light library in Coq. URL: `https://github.com/Deducteam/coq-hol-light`.

16 Frédéric Blanqui, Gilles Dowek, Emilie Grienenberger, Gabriel Hondet, and François Thiré. A modular construction of type theories. *Logical Methods in Computer Science*, Volume 19, Issue 1, February 2023. `doi:10.46298/lmcs-19(1:12)2023`.

17 Valentin Blot, Gilles Dowek, and Thomas Traversié. An Implementation of Set Theory with Pointed Graphs in Dedukti. In *LFMTP 2022 - International Workshop on Logical Frameworks and Meta-Languages : Theory and Practice*, Haïfa, Israel, August 2022. URL: `https://inria.hal.science/hal-03740004`.

18 Denis Cousineau and Gilles Dowek. Embedding pure type systems in the lambda-pi-calculus modulo. In Simona Ronchi Della Rocca, editor, *Typed Lambda Calculi and Applications*, pages 102–117, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

19 Deducteam. An encoding of impredicativity, cumulativity and product covariance in the logical framework dedukti. URL: `https://github.com/Deducteam/cc-in-dk`.

20 Deducteam. Pull request for ACU matching in DkCheck. URL: `https://github.com/Deducteam/Dedukti/pull/219`.

21 Gilles Dowek, Gaspard Férey, Jean-Pierre Jouannaud, and Jiaxiang Liu. Confluence of left-linear higher-order rewrite theories by checking their nested critical pairs. *Mathematical Structures in Computer Science*, 32(7):898–933, 2022. `doi:10.1017/S0960129522000044`.

22 Thiago Felicissimo. Adequate and Computational Encodings in the Logical Framework Dedukti. In Amy P. Felty, editor, *7th International Conference on Formal Structures for Computation and Deduction (FSCD 2022)*, volume 228 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:18, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.FSCD.2022.25`.

23 Thiago Felicissimo. Generic bidirectional typing for dependent type theories. In Stephanie Weirich, editor, *Programming Languages and Systems*, pages 143–170, Cham, 2024. Springer Nature Switzerland.

24 Thiago Felicissimo, Frédéric Blanqui, and Ashish Kumar Barnawal. Translating Proofs from an Impredicative Type System to a Predicative One. In Bartek Klin and Elaine Pimentel, editors, *31st EACSL Annual Conference on Computer Science Logic (CSL 2023)*, volume 252 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 19:1–19:19, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.CSL.2023.19`.

25 Gaspard Férey and Jean-Pierre Jouannaud. Confluence in non-left-linear untyped higher-order rewrite theories. In *Proceedings of the 23rd International Symposium on Principles and Practice of Declarative Programming*, PPDP '21, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3479394.3479403.

26 Gaspard Férey. *Higher-Order Confluence and Universe Embedding in the Logical Framework*. These, Université Paris-Saclay, June 2021. URL: https://tel.archives-ouvertes.fr/tel-03418761.

27 Jan Herman Geuvers. *Logics and type systems*. [Sl: sn], 1993.

28 Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, and Stephan Falke. Automated termination proofs with AProVE. In *Rewriting Techniques and Applications: 15th International Conference, RTA 2004, Aachen, Germany, June 3-5, 2004. Proceedings 15*, pages 210–220. Springer, 2004.

29 Yoan Géran. Mathématiques inversées de Coq, 2021. URL: https://inria.hal.science/hal-04319183.

30 Makoto Hamana. How to prove your calculus is decidable: Practical applications of second-order algebraic theories and computation. *Proc. ACM Program. Lang.*, 1(ICFP), August 2017. doi:10.1145/3110266.

31 Hugo Herbelin. Type inference with algebraic universes in the calculus of inductive constructions. *Unpublished. Available at: pauillac. inria. fr/ herbelin/publis/univalgcci. pdf*, 2005.

32 Gabriel Hondet and Frédéric Blanqui. Encoding of Predicate Subtyping with Proof Irrelevance in the Lambdapi-Calculus Modulo Theory. In Ugo de'Liguoro, Stefano Berardi, and Thorsten Altenkirch, editors, *26th International Conference on Types for Proofs and Programs (TYPES 2020)*, volume 188 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.TYPES.2020.6.

33 Jan Willem Klop. *Combinatory reduction systems*. PhD thesis, Rijksuniversiteit Utrecht, 1963.

34 Marc Lasson. *Réalisabilité et paramétricité dans les systèmes de types purs*. Theses, Ecole normale supérieure de lyon - ENS LYON, November 2012. URL: https://theses.hal.science/tel-00770669.

35 Zhaohui Luo. *An extended calculus of constructions*. PhD thesis, University of Edinburgh, 1990.

36 Richard Mayr and Tobias Nipkow. Higher-order rewrite systems and their confluence. *Theoretical computer science*, 192(1):3–29, 1998.

37 Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of logic and computation*, 1(4):497–536, 1991.

38 Julian Nagele, Bertram Felgenhauer, and Aart Middeldorp. CSI: New evidence — a progress report. In Leonardo de Moura, editor, *Proceedings of the 26th International Conference on Automated Deduction (CADE-26)*, volume 10395 of *Lecture Notes in Artificial Intelligence*, pages 385–397, 2017. doi:10.1007/978-3-319-63046-5_24.

39 Nicolas Oury. Extensionality in the calculus of constructions. In *Theorem Proving in Higher Order Logics: 18th International Conference, TPHOLs 2005, Oxford, UK, August 22-25, 2005. Proceedings 18*, pages 278–293. Springer, 2005.

40 Ronan Saillard. Rewriting modulo β in the $\lambda\pi$-calculus modulo. *Electronic Proceedings in Theoretical Computer Science*, 185:87–101, July 2015. doi:10.4204/eptcs.185.6.

41 Ronan 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.

42 The Coq Development Team. Typing rules for Coq. URL: https://coq.inria.fr/doc/V8.16.1/refman/language/cic.html#id6.

43 René Thiemann and Christian Sternagel. Certification of termination proofs using ceta. In *International Conference on Theorem Proving in Higher Order Logics*, pages 452–468. Springer, 2009.

44 François Thiré. Sharing a library between proof assistants: Reaching out to the HOL family. In Frédéric Blanqui and Giselle Reis, editors, *Proceedings of the 13th International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice, LFMTP@FSCD 2018, Oxford, UK, 7th July 2018*, volume 274 of *EPTCS*, pages 57–71, 2018. doi:10.4204/EPTCS.274.5.

45 François Thiré. *Interoperability between proof systems using the logical framework Dedukti*. PhD thesis, ENS Paris-Saclay, 2020.

46 Vincent van Oostrom. Developing developments. *Theoretical Computer Science*, 175(1):159–181, 1997.

47 Vincent van Oostrom and Femke van Raamsdonk. Weak orthogonality implies confluence: The higher-order case. In Gerhard Goos, Juris Hartmanis, Anil Nerode, and Yu. V. Matiyasevich, editors, *Logical Foundations of Computer Science*, volume 813, pages 379–392. Springer Berlin Heidelberg, Berlin, Heidelberg, 1994. Series Title: Lecture Notes in Computer Science. doi:10.1007/3-540-58140-5_35.

48 Théo Winterhalter, Matthieu Sozeau, and Nicolas Tabareau. Eliminating Reflection from Type Theory. In *CPP 2019 – 8th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 91–103, Lisbonne, Portugal, January 2019. ACM. doi:10.1145/3293880.3294095.

Automating Boundary Filling in Cubical Agda

Maximilian Doré
Department of Computer Science, University of Oxford, United Kingdom

Evan Cavallo
Department of Computer Science and Engineering, University of Gothenburg, Sweden

Anders Mörtberg
Department of Mathematics, Stockholm University, Sweden

Abstract

When working in a proof assistant, automation is key to discharging routine proof goals such as equations between algebraic expressions. Homotopy Type Theory allows the user to reason about higher structures, such as topological spaces, using higher inductive types (HITs) and univalence. `Cubical Agda` is an extension of `Agda` with computational support for HITs and univalence. A difficulty when working in `Cubical Agda` is dealing with the complex combinatorics of higher structures, an infinite-dimensional generalisation of equational reasoning. To solve these higher-dimensional equations consists in constructing cubes with specified boundaries.

We develop a simplified cubical language in which we isolate and study two automation problems: contortion solving, where we attempt to "contort" a cube to fit a given boundary, and the more general Kan solving, where we search for solutions that involve pasting multiple cubes together. Both problems are difficult in the general case – Kan solving is even undecidable – so we focus on heuristics that perform well on practical examples. We provide a solver for the contortion problem using a reformulation of contortions in terms of poset maps, while we solve Kan problems using constraint satisfaction programming. We have implemented our algorithms in an experimental Haskell solver that can be used to automatically solve goals presented by `Cubical Agda`. We illustrate this with a case study establishing the Eckmann-Hilton theorem using our solver, as well as various benchmarks – providing the ground for further study of proof automation in cubical type theories.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Type theory; Computing methodologies → Theorem proving algorithms; Theory of computation → Constraint and logic programming; Theory of computation → Automated reasoning

Keywords and phrases Cubical Agda, Automated Reasoning, Constraint Satisfaction Programming

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.22

Supplementary Material *Software (Source Code)*: `https://github.com/maxdore/dedekind` archived at `swh:1:dir:eb5150105fe3dcbc370e5207f3dcb9de3c26619c`

Funding *Maximilian Doré*: The author was supported by COST Action EuroProofNet, supported by COST (European Cooperation in Science and Technology, `www.cost.eu`).
Evan Cavallo: The author was supported by the Knut and Alice Wallenberg Foundation through the Foundation's program for mathematics.
Anders Mörtberg: The author was supported by the Swedish Research Council (Vetenskapsrådet) under Grant No. 2019-04545.

Acknowledgements We are grateful to Axel Ljungström for discussions about the solver and to him and Tom Jack for cubical versions of Eckmann-Hilton and syllepsis for us to test it with.

1 Introduction

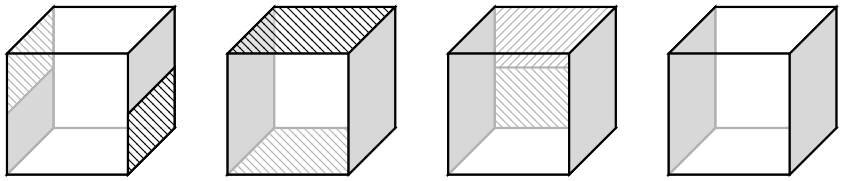
Homotopy Type Theory (HoTT) [35] adds new constructs to intensional dependent type theory [22] reflecting an interpretation of types as homotopy types of topological spaces. This allows homotopy theory to be developed *synthetically* inside HoTT; many classical results have

been reconstructed this way, such as the Hopf fibration [35], Blakers-Massey theorem [16], Seifert-van Kampen theorem [17], Atiyah-Hirzebruch and Serre spectral sequences [36], Hurewicz theorem [8], etc. However, as originally formulated, HoTT postulates both the univalence axiom [38] and the existence of HITs [35] without proper computational content – to rectify this, *cubical type theory* [9] replaces the identity type with a primitive *path* type, yielding a computationally well-behaved theory which validates the axioms of HoTT.

Inspired by Daniel Kan's cubical sets [20], cubical type theory represents elements of iterated identity types as higher-dimensional cubes. Synthetic homotopy theory in cubical type theory thereby attains a particular "cubical" flavour [23]. A path in a type A connecting elements a and b can be thought of as a function $p\colon [\mathbf{0}, \mathbf{1}] \to A$ from the unit interval into the "space" A such that $p(\mathbf{0}) = a$ and $p(\mathbf{1}) = b$. Paths play the role of equalities in the theory, and operations on paths encode familiar laws of equality: reflexivity is a constant path, transitivity is concatenation of paths, and symmetry is following a path in reverse.

Paths can also be studied in their own right. In particular, we can consider equalities *between* paths in A, which as functions $[\mathbf{0}, \mathbf{1}] \to ([\mathbf{0}, \mathbf{1}] \to A)$ can be read as maps from the unit *square* (or *2-cube*) $[\mathbf{0}, \mathbf{1}]^2$ to A; iterating, we find ourselves considering n-cubes in A. Algebraic laws such as the associativity of path concatenation or identity laws are represented as squares with certain boundaries.

For instance, a foundational result in algebraic topology is the *Eckmann-Hilton argument* [12], which states that concatenation of 2-spheres, i.e., 2-cubes with constant boundaries, is commutative up to a path. As a path between 2-cubes, the theorem is a 3-cube as shown in Figure 1(a): on the left we have a gray 2-cube concatenated with a hatched 2-cube, on the right they are concatenated in the opposite order, and the interior is the path between them.



(a) The theorem. **(b)** **(c)** **(d)**

Figure 1 A cubical Eckmann-Hilton argument in four steps.

In cubical type theory, we can construct such an interior by starting from some 3-cube we know can be filled, then deforming its boundary via certain basic operations until it has the desired form. It can be more intuitive to work backwards: deform the "goal" boundary until we reach a boundary we can fill. Figure 1 shows one solution: we (b) shift the copies of the hatched 2-cube to the top and bottom faces, (c) further shift them both to the back face, whereupon they face each other in opposite directions, and then (d) cancel the concatenation of the hatched 2-cube with its inverse. The boundary in (d) can be filled immediately by the constant homotopy – i.e., reflexive equality – from the gray 2-cube to itself.

This example illustrates the two main principles we use to build cubes in type theory, which we call contortion and Kan filling.[1] To *contort* a cube is to reparameterise it, stretching it into a higher dimension or projecting a face. For example, we fill the cube (d) by taking the gray 2-cube and stretching it into a degenerate 3-cube, reparameterizing by a projection

[1] Because we only reason within individual types in this paper, we encounter only so-called *homogeneous* Kan filling. General Kan filling also incorporates *transport* (or *coercion*) between different indices of a dependent type family, but we leave this aspect to future work.

$[\mathbf{0},\mathbf{1}]^3 \to [\mathbf{0},\mathbf{1}]^2$. We cannot hope to derive the cube in Figure 1(a) only by contorting the gray or hatched 2-cube in isolation, of course. Thus the role of *Kan filling*, which lets us modify a cube by a continuous deformation of its boundary. Each of the reductions (a) to (b) to (c) to (d) above is an instance of Kan filling.[2] Kan filling admits a second geometric reading: it states that for every *open box*, i.e. the boundary of a cube with one face unspecified, there is a lid for the box for which an interior ("filler") exists. The two readings agree because a continuous deformation of the boundary of an n-cube over "time" $t \in [\mathbf{0},\mathbf{1}]$ can also be seen as all but two opposing faces of an $(n+1)$-cube; the cube to be deformed fits into one of the missing faces, and the lid produced by box filling is then the deformed cube.

Reasoning with contortions and Kan fillings can pose a challenge when formalising mathematics or computer science in cubical type theory. It is the essence of standalone theorems such as Eckmann-Hilton, but cubical puzzles also often appear as routine lemmas in more complex proofs. One may need to relate one arrangement of concatenations and inverses of paths to another, for example; such coherence conditions often appear in definitions by pattern-matching on HITs. Just as it is difficult to anticipate all types of equations between algebraic expressions that one might need in a large formalisation project, it is infeasible to enumerate every routine cubical lemma in a standard library. The purpose of this paper is to instead devise an algorithm which can automatically prove such lemmas as needed.

We focus on the type theory of `Cubical Agda` [37], currently the most widely used cubical system. The `agda/cubical` [31] and `1lab` [30] libraries each contain ad-hoc collections of cubical reasoning combinators, providing examples on which we test our solver. However, some of the techniques developed in the paper also apply to other cubical type theories, in particular to cartesian cubical type theory [2, 1] as implemented in `redtt` [32] and `cooltt` [33]. A particular feature of `Cubical Agda` that contrasts with cartesian cubical systems is the richer language of contortions, which has proved interesting to study on its own. Like fully-featured cubical type theories, the intended denotational semantics of our language is in cubical sets over any cartesian cube category with connections, following Cohen et al. [9] or Orton and Pitts [24].

Contributions. This paper constitutes one of the first systematic studies of automated reasoning for cubical type theory. In it we

- formulate a minimal cubical language containing a class of "Dedekind" contortions and a Kan filling operator and precisely state the automation problems that we consider (§2),
- formulate an algorithm based on poset maps for solving problems using contortions (§3),
- formulate an algorithm based on constraint satisfaction programming for solving problems using Kan filling (§4), and
- provide a practical `Haskell` implementation of our algorithms and exhibit its effectiveness on a selection of theorems and lemmas taken from libraries for `Cubical Agda` (§5).

2 Boundary problems in cubical type theory

Cubical type theories are complex systems. Besides path types, one has the usual type formers of type theory – functions, products, inductive types, etc. – not to mention univalence and HITs. To make automation tractable, we restrict attention to a fragment including

[2] The fact that a concatenation of a 2-cube with its inverse can be deformed away, which we use in the step (c) to (d), is a lemma that can itself be proven with contortion and Kan filling.

only basic operations on cubical cells in a single type.[3] We also restrict attention to the Dedekind/distributive lattice fragment of `Cubical Agda` and do not rely on the De Morgan involution. In the terminology of `Cubical Agda`, we consider automation for homogeneous Kan filling (hfill) and connections ($\wedge$, $\vee$).

Rather than use path *types* to encode cubical cells, as one does in a fully-featured cubical type theory, we take cells as a primitive notion. A cell is a term parameterised by one or more *dimension variables*, which we think of as ranging in the interval $[\mathbf{0}, \mathbf{1}]$; intuitively, a cell in a type A in n variables is a function $[\mathbf{0}, \mathbf{1}]^n \to A$. Our contexts are lists of cells each of which can have a specified *boundary*. For example, $p(i) : [\, i = \mathbf{0} \mapsto a \mid i = \mathbf{1} \mapsto b \,]$ specifies a 1-dimensional cell p varying in $i \in [\mathbf{0}, \mathbf{1}]$ such that $p(\mathbf{0}) = a$ and $p(\mathbf{1}) = b$, i.e., a path from a to b. In general, a hypothesis has the form $q(\Psi) : [\, \phi \,]$ where Ψ is a list of variables and ϕ is a list of values at faces ($i = \mathbf{0}$ and $i = \mathbf{1}$ in the example above). A cell hypothesis is thus a judgmental analogue of a hypothesis of *extension type* à la Riehl and Shulman [26, §2.2].

The problems we aim to solve are *boundary problems*: given a context of cells Γ, a list of dimension variables Ψ, and a boundary ϕ, can we use the cells in Γ to build a cell varying in Ψ with boundary ϕ? We write such a problem as "$\Gamma \mid \Psi \vdash \mathbf{?} : [\, \phi \,]$". For example, if we want to prove that paths are invertible, then we could pose the boundary problem

$$a : [\,], b : [\,], p(i) : [\, i = \mathbf{0} \mapsto a \mid i = \mathbf{1} \mapsto b \,] \mid j \vdash \mathbf{?} : [\, j = \mathbf{0} \mapsto b \mid j = \mathbf{1} \mapsto a \,] \tag{1}$$

Here Γ has three cells: points a and b, and a path p. Our goal is a path from b to a, written as a function of $j \in [\mathbf{0}, \mathbf{1}]$ with fixed endpoints. We can leave the boundary of cells partially or completely unspecified, so we can formulate the same problem more compactly as

$$p(i) : [\,] \mid j \vdash \mathbf{?} : [\, j = \mathbf{0} \mapsto p(\mathbf{1}) \mid j = \mathbf{1} \mapsto p(\mathbf{0}) \,] \tag{2}$$

Now we assume a path p without naming its endpoints and seek a path from $p(\mathbf{1})$ to $p(\mathbf{0})$. The format extends gracefully to higher cells; the diagonal of a square can be obtained by posing

$$s(i, j) : [\,] \mid k \vdash \mathbf{?} : [\, k = \mathbf{0} \mapsto s(\mathbf{0}, \mathbf{0}) \mid k = \mathbf{1} \mapsto s(\mathbf{1}, \mathbf{1}) \,] \tag{3}$$

Here we assume a 2-dimensional cell s with unspecified boundary and seek a path from $s(\mathbf{0}, \mathbf{0})$ to $s(\mathbf{1}, \mathbf{1})$. In the remaining section, we introduce boundary problems more formally together with the two operations used to build solutions: contortions and Kan filling.

2.1 Boundary problems and contortion solving

The problem (3) has a simple solution: $\mathbf{?} := s(k, k)$. That is, we take the hypothesised 2-cube s and apply a reparameterisation $k \mapsto (k, k)$. We call such reparameterisations *contortions*. In addition to variables and the constants $\mathbf{0}$,$\mathbf{1}$, we follow Cohen et al. (CCHM) [9] and include binary operators $\wedge$ and $\vee$, conventionally called *connections* [5], in our language for contortions. We think of $\wedge$ as taking the *minimum* of two parameters and $\vee$ as taking the maximum; they obey the laws of a bounded distributive lattice with $\mathbf{0}$ and $\mathbf{1}$ as bottom and top element respectively. For example, given a cell context containing a path p, we can define a square whose value at the coordinate (j, k) is the value of p at their maximum:

$$p(i) : [\,] \mid j, k \vdash p(j \vee k) : \left[\begin{array}{lcl} j = \mathbf{0} \mapsto p(k) & \mid & k = \mathbf{0} \mapsto p(j) \\ j = \mathbf{1} \mapsto p(\mathbf{1}) & \mid & k = \mathbf{1} \mapsto p(\mathbf{1}) \end{array}\right] \tag{4}$$

[3] This is similar to the fragments of type theory used to axiomatise higher structures such as weak ω-groupoids in e.g. [6, Appendix A] and [13].

Thus one way to solve a boundary problem is by applying a contortion to a hypothesised cell; searching for such solutions is the first automation problem we consider.

▶ Remark 1. Some cubical type theories, such as that of Angiuli et al. [2, 1], omit connections, while `Cubical Agda` also has the De Morgan involution of CCHM which sends $i \in [\mathbf{0}, \mathbf{1}]$ to $\mathbf{1} - i$. While removing connections would make it trivial to solve contortion goals, we would instead be more reliant on the even harder problem of using Kan filling to solve goals. On the other hand, the poset map representation we use for solving in §3 relies on the absence of the involution. Thus we choose the distributive lattice contortion theory as a happy medium.

Let us now introduce the formal language of boundary problems and solutions.

▶ **Definition 2.** *A **dimension context** Ψ is either a list of (unique) dimension variables $(i_1, \dots, i_n)$ or the **inconsistent context** $\bot$.*

We think of a dimension context with n variables as a unit n-cube, while $\bot$ is the empty space; note that the "empty" context $()$ is the unit 0-cube, which does have a unique point.

▶ **Definition 3.** *A **dimension term** $\Psi \vdash r$ dim over a dimension context Ψ is a term in the free bounded distributive lattice over Ψ. All terms are equal in the inconsistent dimension context. A dimension term r is **atomic** when it is a variable or endpoint, in which case we write $\Psi \vdash r$ atom. We write $\overline{e}$ for the opposite of an endpoint e, so $\overline{\mathbf{0}} \coloneqq \mathbf{1}$ and $\overline{\mathbf{1}} \coloneqq \mathbf{0}$.*

▶ **Definition 4.** *A **contortion** $\psi \colon \Psi' \rightsquigarrow \Psi$ when $\Psi = (i_1, \dots, i_n)$ is a list $\psi = (r_1, \dots, r_n)$ of dimension terms over Ψ'. There is a contortion $\psi \colon \Psi' \rightsquigarrow \bot$ only when $\Psi' = \bot$, in which case there is a unique one. A **substitution** $\psi \colon \Psi' \to \Psi$ is a contortion whose terms are atomic.*

A substitution defines an operation on terms in the usual way: given some kind of term t and a substitution $\psi \colon \Psi' \to \Psi$ where $\Psi = (i_1, \dots, i_n)$ and $\psi = (r_1, \dots, r_n)$, we write $t[\psi]$ for the result of replacing each i_k by r_k in t. Only some of our syntactic sorts are closed under application of general contortions, namely dimension terms and contorted cells (Definition 9); for those sorts we write $t\langle\psi\rangle$ for application of a contortion.

We will need the following operation on dimension contexts to define boundaries.

▶ **Definition 5.** *When Ψ is a dimension context, r is an atomic dimension term, and e is an endpoint, we define the **constrained** dimension context $\Psi[r = e]$ by cases:*

$$(\Psi, i, \Psi')[i = e] \coloneqq \Psi, \Psi' \qquad \Psi[\overline{e} = e] \coloneqq \bot \qquad \Psi[r = e] \coloneqq \Psi, \text{ otherwise}$$

*We have a **constraining substitution** $(r = e) \colon \Psi[r = e] \to \Psi$ that sends r to e if r is a variable, is the unique substitution from $\bot$ when r is $\overline{e}$, and the identity substitution otherwise.*

The *cell contexts* (Γ ctxt), *contorted boundaries* ($\Gamma \mid \Psi \vdash_{\mathrm{c}} \phi$ bdy), and *contorted cells* ($\Gamma \mid \Psi \vdash_{\mathrm{c}} t$ cell and $\Gamma \mid \Psi \vdash_{\mathrm{c}} t : [\,\phi\,]$) are mutually inductively defined as follows.

▶ **Definition 6.** *The **cell contexts** Γ ctxt are inductively defined by rules*

$$\frac{}{() \ \mathsf{ctxt}} \qquad \frac{\Gamma \ \mathsf{ctxt} \qquad \Gamma \mid \Psi \vdash_{\mathrm{c}} \phi \ \mathsf{bdy}}{\Gamma, a(\Psi) : [\,\phi\,] \ \mathsf{ctxt}}$$

That is, a cell context is a list of variables each paired with a dimension context and boundary over that context; the boundary for one variable may mention preceding variables.

▶ **Definition 7.** *The **contorted boundaries** $\Gamma \mid \Psi \vdash_{\mathrm{c}} \phi$ bdy are inductively defined by rules*

$$\frac{}{\Gamma \mid \Psi \vdash_{\mathrm{c}} () \text{ bdy}} \qquad \frac{\Gamma \mid \Psi \vdash_{\mathrm{c}} \phi \text{ bdy} \qquad \Psi \vdash r \text{ atom} \qquad e \in \{\mathbf{0}, \mathbf{1}\} \qquad \Gamma \mid \Psi[r = e] \vdash_{\mathrm{c}} t : [\,\phi[r = e]\,]}{\Gamma \mid \Psi \vdash_{\mathrm{c}} (\phi \mid r = e \mapsto t) \text{ bdy}}$$

Here $\phi[r = e]$ is the application of the constraining substitution $(r = e)$ to the boundary ϕ.

A contorted boundary is thus a list of entries $r = e \mapsto t$, where t is a contorted cell over $\Psi[r = e]$, such that each entry agrees with the previous entries when their constraints overlap.

▶ Remark 8. The requirement that the term r in a constraint $r = e$ is atomic is absent in `Cubical Agda`. Imposing it simplifies the constrained context operation (Definition 5), while relaxing it is not particularly useful for practical boundary solving. We distinguish between substitutions and contortions to make this requirement sensible.

▶ **Definition 9.** *A **contorted cell** $\Gamma \mid \Psi \vdash_{\mathrm{c}} t$ cell is an application $t = a(\psi)$ of a variable $a(\Psi') : [\,\phi\,]$ in Γ to some $\psi \colon \Psi \rightsquigarrow \Psi'$. Equality of contorted cells is generated by the rule*

$$\frac{(a(\Psi') : [\,\phi\,]) \in \Gamma \qquad (r = e \mapsto t) \in \phi \qquad \psi \colon \Psi \rightsquigarrow \Psi' \qquad \Psi \vdash r\langle\psi\rangle = e\langle\psi\rangle \text{ dim}}{\Gamma \mid \Psi \vdash_{\mathrm{c}} a(\psi) = t\langle\psi\rangle \text{ cell}}$$

which is to say that a has the boundary assigned in the context. We write $\Gamma \mid \Psi \vdash_{\mathrm{c}} t : [\,\phi\,]$ when t is a cell agreeing with ϕ, i.e., such that $t[r = e] = t'$ cell for each $(r = e \mapsto t') \in \phi$.

▶ Remark 10. We normalise a contorted cell by looking at its boundary, if it is specified. For example, in context (1) the cell $p(\mathbf{0})$ normalises to a. The number of steps necessary to normalise a contorted cell is bounded by the length of the context.

We can now state the first of the two problems we are interested in solving.

▶ **Problem 11** (Contortion)**.** *Given $\Gamma \mid \Psi \vdash_{\mathrm{c}} \phi$ bdy, the problem Contortion(Γ, Ψ, ϕ) is to determine if there exists a contorted cell t such that $\Gamma \mid \Psi \vdash_{\mathrm{c}} t : [\,\phi\,]$.*

The problem Contortion(Γ, Ψ, ϕ) is decidable: there are finitely many cell variables in Γ, so we could try all possible contortions of each by brute force. On the other hand, it is certainly not easily solved efficiently. Even restricting to 1-dimensional goals, Contortion is NP-hard when the contortion language includes connections.

▶ **Proposition 12.** *Contortion(Γ, Ψ, ϕ) is NP-complete for Ψ with one variable.*

Proof. First, note that this problem is in NP: when the cardinality of Ψ is fixed, the normalisation necessary to verify a putative solution can be done in polynomial time. For completeness, we give a reduction from SAT. Suppose we have a Boolean CNF formula φ over $\vec{x} = x_1, \ldots, x_n$. Replace each $\neg x_i$ in φ by a variable y_i to obtain a dimension term r in variables $\vec{x}, \vec{y}$. Then φ is satisfiable if and only if there is $\psi \colon () \rightsquigarrow (\vec{x}, \vec{y})$ such that $r\langle\psi\rangle = \mathbf{1}$ and $(x_k \wedge y_k)\langle\psi\rangle = \mathbf{0}$ and $(x_k \vee y_k)\langle\psi\rangle = \mathbf{1}$ for each k. Take Γ_φ to be the context

$$a : [\,], p(z, j_0, j_1) : [\,], q(\vec{x}, \vec{y}, i) : [\,i = \mathbf{0} \mapsto a \mid i = \mathbf{1} \mapsto p\,(r, \textstyle\bigvee_k (x_k \wedge y_k), \bigwedge_k (x_k \vee y_k))\,]$$

and consider the boundary problem $\Gamma_\varphi \mid i \vdash_{\mathrm{c}} \mathbf{?} : [\,i = \mathbf{0} \mapsto a \mid i = \mathbf{1} \mapsto p(\mathbf{1}, \mathbf{0}, \mathbf{1})\,]$. Any $\psi \colon () \rightsquigarrow (\vec{x}, \vec{y})$ such that $r\langle\psi\rangle = \mathbf{1}$ and $(x_k \wedge y_k)\langle\psi\rangle = \mathbf{0}$ and $(x_k \vee y_k)\langle\psi\rangle = \mathbf{1}$ for each k yields a solution $\Gamma_\varphi \mid i \vdash_{\mathrm{c}} q(\psi, i)$ cell. Conversely, any solution to the problem will be of the form $\Gamma_\varphi \mid i \vdash_{\mathrm{c}} q(\psi', r)$ cell for some $\psi' \colon i \rightsquigarrow (\vec{x}, \vec{y})$ and $i \vdash r$ dim, in which case $\psi'(\mathbf{1}) \colon () \rightsquigarrow (\vec{x}, \vec{y})$ induces a satisfying assignment for φ. ◀

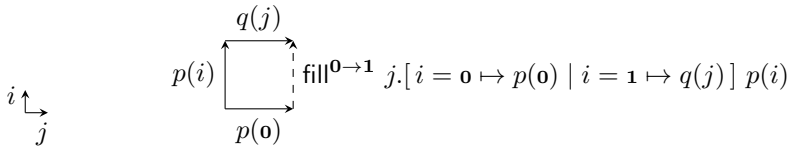
We expect that Contortion is significantly *more* complex in general and that the complexity is dominated by the dimensionality of the goal. The number of ways to contort an m-cube to fit an n-dimensional goal is $D(n)^m$ where $D(n)$ is the n-th *Dedekind number* [4, App. B]. While the exponential dependence on m is already problematic, the Dedekind numbers grow even more quickly. There are $D(6) = 7\,828\,354$ many ways to contort a 1-cube into a 6-dimensional cube; the 42-digit $D(9)$ was only recently computed using supercomputing [15, 19]. Thus, our focus in this paper is on heuristics that quickly yield solutions to boundary problems that appear in practice, rather than on worst-case asymptotics.

2.2 Kan filling

Paths in spaces can be *concatenated*: if there is a path from a to b and a path from b to c, then there is a path from a to c. Concatenation generalises to higher cells; for example, we can attach several surfaces at their boundaries to form a new surface. Kan [20] devised a single property that encompasses all of these operations in the context of cubical sets. In cubical type theory, it is embodied by the *Kan filling* (sometimes *Kan composition*) operator.

We write an application of the filling operator as $\mathsf{fill}^{e\to r}\ j.[\,\phi\,]\ u$, where u is a cell, $j.[\,\phi\,]$ is a boundary varying in a dimension variable j, e is an endpoint, and r is an atomic dimension term. For the operator to be well-formed u must have boundary $\phi[j \mapsto e]$, while the resulting cell has boundary $\phi[j \mapsto r]$; thus we think of fill as deforming the boundary of u from $\phi[j \mapsto e]$ to $\phi[j \mapsto r]$. The fact that we fill to a dimension term r means that the operation unifies hcomp/hfill of [10], while being a special case of the more general hcom of [2, 1].

As an example, consider the context $p(i) : [\,], q(j) : [\,j = \mathbf{0} \mapsto p(\mathbf{1})\,]$ with p and q such that the $\mathbf{1}$-endpoint of p lines up with the $\mathbf{0}$-endpoint of q. Suppose we want to concatenate them and produce a cell with boundary $[\,i = \mathbf{0} \mapsto p(\mathbf{0}) \mid i = \mathbf{1} \mapsto q(\mathbf{1})\,]$. Observe that the boundary $[\,i = \mathbf{0} \mapsto p(\mathbf{0}) \mid i = \mathbf{1} \mapsto q(j)\,]$ varying in j is the boundary of p at "time" $j = \mathbf{0}$ and of our desired concatenation at "time" $j = \mathbf{1}$. Thus, deforming p with fill can give us our goal:



We write $(p \cdot q)(i)$ for the filler above. If we replace the target $\mathbf{1}$ of the fill with a variable, we get an interior for the depicted square. One says that this cell *fills* the open box formed by the solid paths, hence the name. In general, we have the following rules for building Kan cells:

▶ **Definition 13.** *The* ***Kan cells*** $\Gamma \mid \Psi \vdash t\ \mathsf{cell}$ *are inductively generated by the rules*

$$\frac{\Gamma \mid \Psi \vdash_{\mathrm{c}} t\ \mathsf{cell}}{\Gamma \mid \Psi \vdash t\ \mathsf{cell}} \qquad \frac{e \in \{\mathbf{0}, \mathbf{1}\} \qquad \Psi \vdash r\ \mathsf{atom} \qquad \Gamma \mid \Psi, i \vdash \phi\ \mathsf{bdy} \qquad \Gamma \mid \Psi \vdash u : [\,\phi[i \mapsto e]\,]}{\Gamma \mid \Psi \vdash \mathsf{fill}^{e\to r}\ i.[\,\phi\,]\ u : [\,\phi[i \mapsto r], r = e \mapsto u\,]}$$

The ***Kan boundaries*** $\Gamma \mid \Psi \vdash \phi\ \mathsf{bdy}$ *are defined analogously to the contorted boundaries* $\Gamma \mid \Psi \vdash_{\mathrm{c}} \phi\ \mathsf{bdy}$*, and we write* $\Gamma \mid \Psi \vdash t : [\,\phi\,]$ *to mean that* t *is a Kan cell agreeing with* ϕ*.*

▶ Remark 14. Note that the $r = e \mapsto u$ constraint makes filling in direction $e \to e$ the identity function. This ensures that the face opposite the missing side in the filler agrees with the input u to the filling operator.

With this we can now state the second of the two problems we are interested in solving.

▶ **Problem 15** (KAN). *Given a Kan boundary* $\Gamma \mid \Psi \vdash \phi$ bdy*, the problem* $\textsc{Kan}(\Gamma, \Psi, \phi)$ *is to determine if there exists a Kan cell* t *such that* $\Gamma \mid \Psi \vdash t : [\,\phi\,]$.

For example, the problem (2) of inverting a path does not have a solution in CONTORTION but does have solutions in KAN, such as $\mathsf{fill}^{\mathbf{0}\to\mathbf{1}}\ i.[\,j = \mathbf{0} \mapsto p(i) \mid j = \mathbf{1} \mapsto p(\mathbf{0})\,]\ p(\mathbf{0})$.

▶ Remark 16. We claim that Kan is undecidable, but give only a proof sketch here. Although the main idea is simple, a fully precise proof would be combinatorially involved. We can encode the word problem for a finitely presented group G as an instance of KAN. Such a group can always be presented by a finite set of generators X and relations $ab = c$ where $a, b, c \in X$. Take the context Γ with a point $\star : [\,]$, a loop $\hat{a}(i) : [\,i = \mathbf{0} \mapsto \star \mid i = \mathbf{1} \mapsto \star\,]$ for each $a \in X$, and a square $s_{abc}(j,k) : [\,j = \mathbf{0} \mapsto \hat{a}(k) \mid j = \mathbf{1} \mapsto \star \mid k = \mathbf{0} \mapsto \hat{c}(j) \mid k = \mathbf{1} \mapsto \hat{b}(j)\,]$ for each $ab = c$. Note that each square s_{abc} expresses that the composite of the loops $\hat{a}$ and $\hat{b}$ is equal up to a path to the loop $\hat{c}$. Given arbitrary words w, v over X, we have loops $\Gamma \mid i \vdash \hat{w}, \hat{v} : [\,i = \mathbf{0} \mapsto \star \mid i = \mathbf{1} \mapsto \star\,]$ defined by concatenating generator loops. We claim that $w = v$ in G if and only if $\Gamma \mid i, \ell \vdash \mathbf{?} : [\,i = \mathbf{0} \mapsto \star \mid i = \mathbf{1} \mapsto \star \mid \ell = \mathbf{0} \mapsto \hat{w} \mid \ell = \mathbf{1} \mapsto \hat{v}\,]$ has a solution in KAN, i.e., if $\hat{w}$ and $\hat{v}$ are equal up to a path.

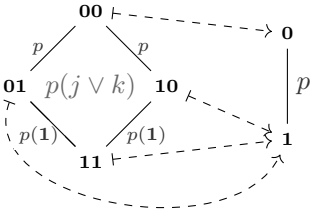
3 Finding contortions

The key to our approach for solving the CONTORTION problem is a characterisation of contortions as poset maps by Stone duality [18]. In §3.1, we introduce this characterisation along with *potential poset maps (PPMs)*, a lossy but space-saving representation of collections of poset maps. We use PPMs in §3.2 to develop an algorithm for solving CONTORTION.

3.1 Representing contortions with potential poset maps

Recall the example (4), where we contorted a path p into a square $p(i) : [\,] \mid j, k \vdash p(j \vee k)$ cell. We can think of $\vee$ as logical disjunction – if either j or k is $\mathbf{1}$, the contortion evaluates to $\mathbf{1}$. Similarly, we can treat the connection $\wedge$ as logical conjunction, which means that we can view any contortion as a tuple of propositional formulas. In particular, every contortion has a corresponding truth table, and in fact a contortion is uniquely determined by its truth table; for example, the contortion above is determined by the assignment $[\![-]\!] \colon \{\mathbf{0},\mathbf{1}\} \times \{\mathbf{0},\mathbf{1}\} \to \{\mathbf{0},\mathbf{1}\}$ defined by $[\![\mathbf{00}]\!] = \mathbf{0}$ and $[\![\mathbf{01}]\!] = [\![\mathbf{10}]\!] = [\![\mathbf{11}]\!] = \mathbf{1}$. In general, an n-term contortion in m variables gives a truth function $\{\mathbf{0},\mathbf{1}\}^m \to \{\mathbf{0},\mathbf{1}\}^n$.

Since a contortion ψ contains no negations, its truth function is *monotone* – we cannot make ψ false by setting more variables to true. Thus the truth function induced by ψ is in fact a map of posets $\mathbf{I}^m \to \mathbf{I}^n$, where $\mathbf{I}^k$ is the k-fold power of the poset $\mathbf{I} := \{\mathbf{0} < \mathbf{1}\}$ with its product ordering. Conversely, any map of posets $\mathbf{I}^m \to \mathbf{I}^n$ determines a unique n-term contortion in m variables. For example, we can depict the poset map corresponding to $j \vee k$ as an assignment between the posets $\mathbf{I}^2$ and $\mathbf{I}^1$, which we draw as a Hasse diagram:



We can read off the boundary of $p(j \vee k)$ by looking at the action of the poset map: the edge from $\mathbf{00}$ to $\mathbf{01}$ is sent to the edge from $\mathbf{0}$ to $\mathbf{1}$ in the target, so the $j = \mathbf{0}$ side of $p(j \vee k)$ is $p(k)$. Between $\mathbf{01}$ and $\mathbf{11}$, we stay at $\mathbf{1}$, so the boundary at $j = \mathbf{1}$ is constantly $p(\mathbf{1})$.

We will in the following freely switch between regarding contortions as tuples of propositional formulas and as poset maps. We write $\psi_{\mathbf{I}}$ for the poset map induced by the contortion ψ and $\sigma_{\vee\wedge}$ for the contortion prescribed by the poset map σ. The poset map perspective on contortions does not only give geometric intuition for the boundary of a contorted cell, but also allows for concisely representing a collection of contortions: by assigning a *set* of values $Y \subseteq \mathbf{I}^n$ to each element of $\mathbf{I}^m$, we can at once represent several contortions. The monotonicity constraint on poset maps entails that only some such assignments are meaningful; we call these *potential poset maps*.

▶ **Definition 17.** *A **potential poset map (PPM)** is a map $\Sigma\colon \mathbf{I}^m \to \mathcal{P}(\mathbf{I}^n)$ s.t. $\forall x \leq y$:*

$$\forall u \in \Sigma(y).\ \exists v \in \Sigma(x).\ v \leq u \qquad \textit{and} \qquad \forall v \in \Sigma(x).\ \exists u \in \Sigma(y).\ v \leq u$$

With a PPM, we can represent a collection of contortions with very little data: representing all $D(m)^n$ poset maps $\mathbf{I}^m \to \mathbf{I}^n$ with the total PPM $x \mapsto \mathbf{I}^n$ for $x \in \mathbf{I}^m$ requires 2^m entries of 2^n values – the memory requirements are therefore independent of the Dedekind numbers and grow "only" exponentially in m and n. This comes with the trade-off that PPMs are a lossy representation of sets of poset maps. For example, any PPM containing the two poset maps $\sigma, \sigma' : \mathbf{I}^1 \to \mathbf{I}^2$ defined by $\sigma(\mathbf{0}) = (\mathbf{00}), \sigma(\mathbf{1}) = (\mathbf{10})$ and $\sigma'(\mathbf{0}) = (\mathbf{01}), \sigma'(\mathbf{1}) = (\mathbf{11})$ also contains the diagonal map sending $\mathbf{0} \mapsto (\mathbf{00})$ and $\mathbf{1} \mapsto (\mathbf{11})$.

In the following, we unfold a PPM Σ into the set of poset maps it contains with UnfoldPPM(Σ). We update a PPM Σ to restrict its values at x to some set $vs \subseteq \Sigma(x)$ using UpdatePPM(Σ, x, vs), thereby obtaining a new PPM Σ' with $\Sigma'(x) = vs$. Due to space constraints we refer to the source code of the solver discussed in §5 for details.

3.2 An algorithm for gradually constructing contortions

We now use PPMs in Algorithm 1 to solve Contortion more efficiently than by brute force. Given a boundary problem $\Gamma \mid \Psi \vdash_{\mathrm{c}} \mathbf{?} : [\phi]$ and a cell $a(\Psi') : [\phi']$ in Γ, we search for a contortion $\psi\colon \Psi \rightsquigarrow \Psi'$ such that $a\langle\psi\rangle$ has boundary ϕ by gradually restricting a PPM to be compatible with the faces of ϕ. If a contortion of a appears as one of the faces of ϕ, we can even reduce the search space significantly before performing any expensive operations.

We first initialize Σ to the total PPM on line 2. We then go through the faces of ϕ, each of which normalises to the form $b\langle\psi\rangle$ for some variable b and contortion ψ, and use them to restrict Σ. Crucially, we order the boundary faces by descending dimensionality of the contorted variable on line 3, as contortions of higher-dimensional variables constrain the search space more. Given a face $i = e \mapsto b\langle\psi\rangle$ of ϕ, we proceed as follows: if b is in fact a, we can constrain Σ to maps that agree with ψ where $i = e$ on line 5. Otherwise, we iterate through the poset maps σ contained in the restriction of Σ to $\mathbf{I}^m_{i=e}$ on line 8. For each, we mark its values for retention only when $a\langle\sigma_{\vee\wedge}\rangle$ matches the face $b\langle\psi\rangle$. Finally, we propagate our findings to Σ on line 13. After restricting Σ according to all faces of ϕ, we unfold Σ and brute-force search the results for a valid solution to return. Note that not all poset maps in Σ need be solutions, as a PPM is a lossy representation of a set of maps. The algorithm is complete: if a can be contorted by some ψ to solve the goal boundary, then it keeps $\psi_{\mathbf{I}}(x)$ in $\Sigma(x)$ for all $x \in \mathbf{I}^m$ in each iteration of the main loop, whether in line 5 or line 11.

The main expense in Algorithm 1 is unfolding all poset maps from a subposet on line 8. For an unconstrained PPM, we have to check $D(m-1)^n$ poset maps, and as we are doing this for up to $2m$ faces of ϕ, we are unfolding $2m \cdot D(m-1)^n$ poset maps in the worst case. In many boundary problems, the cell to be contorted appears in the boundary, which means the search space significantly shrinks before any PPM is unfolded. This allows us to compute many contortions that would have been impossible to find by naïve brute-force.

Algorithm 1 Constructing a contortion.

Input: $\Gamma \mid \Psi \vdash_{\mathrm{c}} \phi$ bdy and $a(\Psi') : [\,\phi'\,] \in \Gamma$. Let $m \coloneqq |\Psi|$ and $n \coloneqq |\Psi'|$.
Output: $\psi \colon \Psi \rightsquigarrow \Psi'$ s.t. $\Gamma \mid \Psi \vdash_{\mathrm{c}} a\langle\psi\rangle : [\,\phi\,]$ if such a ψ exists, `Unsolvable` otherwise

1: **procedure** CONTORT(Γ, Ψ, ϕ, a)
2: $\quad\Sigma \coloneqq \{x \mapsto \mathbf{I}^n \mid x \in \mathbf{I}^m\}$
3: $\quad$**for** $(i = e \mapsto b\langle\psi\rangle) \in \phi$ with $\psi \colon \Psi[i = e] \rightsquigarrow \Psi''$, in descending order of $|\Psi''|$ **do**
4: $\quad\quad$**if** $a = b$ **then**
5: $\quad\quad\quad\Theta \coloneqq \{x \mapsto \{\psi_{\mathbf{I}}(x)\} \mid x \in \mathbf{I}^m_{i=e}\}$
6: $\quad\quad$**else**
7: $\quad\quad\quad\Theta \coloneqq \{x \mapsto \emptyset \mid x \in \mathbf{I}^m_{i=e}\}$
8: $\quad\quad\quad$**for** $\sigma \in$ UNFOLDPPM$(\Sigma|_{\mathbf{I}^m_{i=e}})$ **do**
9: $\quad\quad\quad\quad$**if** $a\langle\sigma_{\vee\wedge}\rangle = b\langle\psi\rangle$ **then**
10: $\quad\quad\quad\quad\quad$**for** $x \in \mathbf{I}^m_{i=e}$ **do**
11: $\quad\quad\quad\quad\quad\quad\Theta(x) \coloneqq \Theta(x) \cup \{\sigma(x)\}$
12: $\quad\quad$**for** $x \in \mathbf{I}^m_{i=e}$ **do**
13: $\quad\quad\quad$UPDATEPPM$(\Sigma, x, \Theta(x))$
14: $\quad$**if** $\exists\sigma \in$ UNFOLDPPM(Σ) such that $\Gamma \mid \Psi \vdash_{\mathrm{c}} a\langle\sigma_{\vee\wedge}\rangle : [\,\phi\,]$ **then**
15: $\quad\quad$**return** $\sigma_{\vee\wedge}$
16: $\quad$**else**
17: $\quad\quad$**return** `Unsolvable`

▶ **Example 18** (Square to cube contortion)**.** Suppose that we are given the cell context $\Gamma \coloneqq a : [\,], s(i,j) : [\,i = \mathbf{0} \mapsto a \mid i = \mathbf{1} \mapsto a \mid j = \mathbf{0} \mapsto a \mid j = \mathbf{1} \mapsto a\,]$ and want to contort the square s to match the following 3-cube boundary, which has a contortion of s on one face and squares which are constantly a otherwise:

$$\Gamma \mid i,j,k \vdash_{\mathrm{c}} \mathbf{?} : \begin{bmatrix} i = \mathbf{0} \mapsto s(j \wedge k, j \vee k) & \mid & j = \mathbf{0} \mapsto a & \mid & k = \mathbf{0} \mapsto a \\ i = \mathbf{1} \mapsto a & \mid & j = \mathbf{1} \mapsto a & \mid & k = \mathbf{1} \mapsto a \end{bmatrix}$$

This is a difficult instance of CONTORTION because most faces of the goal are contortions of a 0-cell, which can be obtained in many ways. To construct $\psi \colon (i,j,k) \rightsquigarrow (i,j)$ such that $s\langle\psi\rangle$ has boundary ϕ, we search for the equivalent poset map $\mathbf{I}^3 \to \mathbf{I}^2$ using Algorithm 1.

On line 2, the total PPM $\Sigma : \mathbf{I}^3 \to \mathcal{P}(\mathbf{I}^2)$ is initialized with $x \mapsto \mathbf{I}^2$ for all $x \in \mathbf{I}^3$. We then go through all faces of the goal boundary and use them to restrict Σ, starting with the contortion of s at $i = \mathbf{0}$. Since s is also the cell that we are contorting, the subposet $\mathbf{I}^3_{i=\mathbf{0}}$ of the domain of Σ is mapped in a unique way to the elements of $\mathbf{I}^2$. The monotonicity restrictions on PPMs further restrict Σ, which only contains 10 poset maps after this first restriction. In the next iteration of the outer loop, we only have degenerate a faces left in the goal boundary. Going through each face further restricts Σ, as most induced poset maps give rise to a contortion of s which is not the constant a square. Afterwards, Σ comprises a single poset map: $\Sigma(\mathbf{000}) = \{\mathbf{00}\}$, $\Sigma(\mathbf{001}) = \Sigma(\mathbf{010}) = \Sigma(\mathbf{011}) = \Sigma(\mathbf{100}) = \Sigma(\mathbf{101}) = \{\mathbf{01}\}$ and $\Sigma(\mathbf{110}) = \Sigma(\mathbf{111}) = \{\mathbf{11}\}$. Translating this poset map to a contortion gives rise to a solution for our boundary problem: $\Gamma \mid i,j,k \vdash_{\mathrm{c}} s(i \wedge j, i \vee j \vee k) : [\,\phi\,]$

Our algorithm finds this solution quickly since the search space is restricted to only 10 possible contortions after looking at the first face of ϕ. This contrasts with brute-force search, where we would have to check $D(3)^2 = 400$ contortions. The increase in speed gets apparent for a larger goal: a 6-dimensional analogue of the above proof goal can be found by unfolding less than 16000 poset maps. A brute-force search would have to find a solution in a search space with $D(6)^2 = 7\,828\,354^2 = 61\,283\,126\,349\,316$ contortions.

4 Finding Kan fillers

We now turn to KAN and develop an algorithm for solving general boundary problems. Recall that a Kan cell is of the form $\mathsf{fill}^{e \to r}\ i.[\,\phi\,]\ u$, where ϕ and u constitute an "open box" which is filled in direction $e \to r$. Searching for such fillers requires a different approach depending on whether r is a dimension variable or an endpoint. In the former case, $\mathsf{fill}^{e \to j}\ i.[\,\phi\,]\ u$ has the same dimension as ϕ and has $\mathsf{fill}^{e \to \overline{e}}\ i.[\,\phi\,]\ u$ as its $j = \overline{e}$ face. This means that it is easy to recognise if a boundary problem can be solved by a filler $e \to j$: we simply have to check if some face of the goal boundary is an $e \to \overline{e}$ filler. We hence call the filler in direction $e \to j$ the "natural filler" for a goal boundary which has $\mathsf{fill}^{e \to \overline{e}}\ i.[\,\phi\,]\ u$ at side $j = \overline{e}$.

In contrast, determining when we have to introduce $e \to \overline{e}$ fillers is difficult. We focus our attention on fillers in direction $\mathbf{0} \to \mathbf{1}$, since such a filler can be constructed if and only if we can construct a filler in the converse direction. Note that a cell $\mathsf{fill}^{\mathbf{0} \to \mathbf{1}}\ i.[\,\phi\,]\ u$ is of one dimension less than the open box spanned by ϕ and u – put differently, to solve a given goal boundary by a $\mathbf{0} \to \mathbf{1}$ filler, we need to first construct a higher-dimensional cube. We hence call fillers in direction $\mathbf{0} \to \mathbf{1}$ "higher-dimensional" fillers. Searching for such cells is difficult because the goal boundary only partially constrains the faces of ϕ, while u could be any cell of the correct dimension. In particular, we could again use higher-dimensional fillers as faces for ϕ or u, leading to infinite search spaces that we have to carefully navigate with heuristics.

In our solver we follow the principle that when solving a boundary problem, it's best to use contorted cells and natural fillers if possible, and only construct higher-dimensional fillers if necessary. We formulate the problem of finding a higher-dimensional filler which only uses contorted cells as a constraint satisfaction problem (CSP) [21, 34] in §4.1, which allows us to employ finite domain constraint solvers for this sub-problem of KAN. By carefully calling this solver, we then give a complete search procedure for KAN in §4.2.

4.1 Kan filling as a constraint satisfaction problem

When constructing a higher-dimensional cell for a goal boundary, the sides of the open box that we fill need to match up. This suggests a recipe for constructing fillers: we formulate the search problem as a CSP. In this section, we focus on the problem where all the sides of the filler are contortions. Since there are only finitely many contortions into a given dimension, we can use a finite domain constraint solver to solve this CSP. Still, the number of contortions grows very quickly, making it quickly infeasible to list all contortions. To rectify this, we again rely on PPMs. By representing a collection of contortions with a PPM, we can quickly construct our CSP with little memory requirements; a solver such as the one discussed in §5 can then gradually narrow down the PPMs until it arrives at a solution.

Recall that a CSP is given by a set of variables Var; an assignment of domains to Var, i.e., a set D_X for each $X \in Var$; and a set of constraints $C \subseteq D_X \times D_{X'}$ for $X, X' \in Var$. A solution is a choice of one element of each domain, i.e., $t_X \in D_X$ for all $X \in Var$, s.t., all constraints are satisfied, i.e., $C(t_X, t_{X'})$ for all C, X, X'. We now state the CSP for filling boundaries via Kan fillers that have only contortions as sides.

▶ **Definition 19.** *Given a boundary* $\Gamma \mid \Psi \vdash \phi\ \mathsf{bdy}$ *and a fresh dimension* $k \notin \Psi$*, as well as a set of indices* $Ope \subseteq \{(k = \mathbf{0})\} \cup \{(i = e) \mid i \in \Psi, e \in \{\mathbf{0}, \mathbf{1}\}\}$*, the CSP* $\textsc{KanCSP}(\phi, Ope)$ *is given as follows:*

- $Var := \{X_{(i=e)} \mid i \in \Psi, e \in \{\mathbf{0}, \mathbf{1}\}, (i = e) \notin Ope\} \cup \{X_{(k=\mathbf{0})} \text{ if } (k = \mathbf{0}) \notin Ope\}$
- $D_{(i=e)} := \{(p, \Sigma) \mid p(\Psi') : [\,\ldots\,] \in \Gamma, \Sigma : \mathbf{I}^{|\Psi|} \to \mathcal{P}(\mathbf{I}^{|\Psi'|})\}$

- *and constraints for all* $\Psi \vdash i, j$ atom*,* $e, e' \in \{\mathbf{0}, \mathbf{1}\}$*:*

$$\Gamma \mid \Psi[i = e] \vdash_{\mathrm{c}} X_{(i=e)}[k = \mathbf{1}] = \phi[i = e]\ \mathsf{cell}\ \textit{if}\ (i, e)\ \textit{specified in}\ \phi$$
$$\Gamma \mid \Psi[i = e][j = e'] \vdash_{\mathrm{c}} X_{(i=e)}[j = e'] = X_{(j=e')}[i = e]\ \mathsf{cell}$$

The CSP contains a variable for any side of the boundary that is not left open, the domains contain all pairs of a cell p and PPM Σ contorting p into the right dimension. The first set of constraints ensures that all sides agree with the goal boundary, while the second set of constraints makes sure that all sides have mutually matching boundaries.

If *Ope* contains only sides which are unspecified in ϕ, a solution $\textsc{KanCSP}(\phi, \mathit{Ope})$ is a solution to the boundary problem ϕ:

$$\Gamma \mid \Psi \vdash \mathsf{fill}^{\mathbf{0}\to\mathbf{1}}\ k.[\, i = e \mapsto t_{(i,e)} \text{ for } i \in \Psi, e \in \{\mathbf{0}, \mathbf{1}\}, (i = e) \notin \mathit{Ope}\,]\ t_{(k,\mathbf{0})} : [\,\phi\,]$$

▶ **Example 20** (The Eckmann-Hilton cube)**.** We want to fill the cube from Figure 1(b), for which are given a cell context Γ with a point $x : [\,]$ and two squares $p(i, j)$ and $q(i, j)$ with boundaries $[i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto x \mid j = \mathbf{0} \mapsto x \mid j = \mathbf{1} \mapsto x]$, and which are assembled into:

$$\Gamma \mid i, j, k \vdash \begin{bmatrix} i = \mathbf{0} \mapsto p(j,k) \mid & j = \mathbf{0} \mapsto q(i,k) \mid & k = \mathbf{0} \mapsto x \\ i = \mathbf{1} \mapsto p(j,k) \mid & j = \mathbf{1} \mapsto q(i,k) \mid & k = \mathbf{1} \mapsto x \end{bmatrix} \mathsf{bdy}$$

We try to solve KanCSP with no open sides. This CSP has 7 variables corresponding to sides i, j, k and a backside $l = \mathbf{0}$. After imposing the first set of constraints, the domains for the i and j sides are significantly reduced, e.g., $D_{(i=\mathbf{0})} = \{p\langle\Sigma\rangle\}$ for $\Sigma : \mathbf{I}^3 \to \mathcal{P}(\mathbf{I}^2)$ given by:

$$\begin{array}{llll} \mathbf{000} \mapsto \{\mathbf{00}\} & \mathbf{001} \mapsto \{\mathbf{00}\} & \mathbf{010} \mapsto \{\mathbf{00}, \mathbf{01}\} & \mathbf{011} \mapsto \{\mathbf{01}\} \\ \mathbf{100} \mapsto \{\mathbf{00}, \mathbf{10}\} & \mathbf{101} \mapsto \{\mathbf{10}\} & \mathbf{110} \mapsto \{\mathbf{00}, \mathbf{01}, \mathbf{10}, \mathbf{11}\} & \mathbf{111} \mapsto \{\mathbf{11}\} \end{array}$$

The PPM Σ gives rise to 9 contortions of p, which contrasts with $D(3)^2 = 400$ total contortions of p. The domains for $D_{(k=\mathbf{0})}$, $D_{(k=\mathbf{1})}$, and the back side $D_{(l=\mathbf{0})}$ still contain all contortions of x, p and q into three dimensions since the k sides of the goal boundary does not give any indication which contortion could be used for this side of the filler.

The second set of constraints ensures that all sides of the Kan filler have matching boundaries, after which we find a solution to KanCSP that gives rise to the following filler:

$$\Gamma \mid i, j, k \vdash \mathsf{fill}^{\mathbf{0}\to\mathbf{1}}\ l. \begin{bmatrix} i = \mathbf{0} \mapsto p(j, k \wedge l) & j = \mathbf{0} \mapsto q(i,k) & k = \mathbf{0} \mapsto x \\ i = \mathbf{1} \mapsto p(j, k \wedge l) & j = \mathbf{1} \mapsto q(i,k) & k = \mathbf{1} \mapsto p(j,l) \end{bmatrix} q(i,k)\ \mathsf{cell}$$

This filler captures the argument sketched in Figure 1, albeit in a single step: the p sides are mapped to the $k = \mathbf{1}$ side such that they cancel out as in Figure 1(c), while the q sides are constantly mapped to the backside of the filler, which is the cube from Figure 1(d).

4.2 A solver for Kan

We now give an algorithm to construct fillers of open cubes which might have fillers on their faces, and not only contorted terms as in KanCSP. We can straightforwardly devise a procedure $\textsc{KanFill}(\Gamma, \Psi, \phi)$ which produces fillers with the same dimension as ϕ: we check for any face of ϕ if it gives rise to a natural filler.

The difficult part of Kan is the construction of higher-dimensional fillers, which might possibly have fillers on their sides. We introduce a variable d to iteratively deepen the level of such nested fillers, which effects a sort-of "breadth-first" search for nested fillers.

Algorithm 2 Finding Kan cells.

Input: $\Gamma \mid \Psi \vdash \phi$ bdy, depth variable d
Output: $\Gamma \mid \Psi \vdash t : [\,\phi\,]$, if $\textsc{Kan}(\Gamma, \Psi, \phi)$ solvable with $\leq d$ nested Kan fillers

1: **procedure** $\textsc{KanSolver}(\Gamma, \Psi, \phi, d)$
2: **if** $d = 0$ **then**
3: **return** `Unsolvable`
4: $t \leftarrow \textsc{KanFill}(\Gamma, \Psi, \phi) \cup \textsc{KanCube}(\Gamma, \Psi, \phi, d)$
5: **procedure** $\textsc{KanCube}(\Gamma, \Psi, \phi, d)$
6: $Ope \leftarrow \mathcal{P}(\{(i = e) \mid i \in \Psi, e \in \{\mathbf{0}, \mathbf{1}\}\} \cup \{(k = \mathbf{0})\})$
7: $\phi' \leftarrow \textsc{KanCSP}(\phi, Ope)$
8: **for** $(i = e) \in Ope$ **do**
9: $t \leftarrow \textsc{KanSolver}([\,\phi'[i = e]\,], d - 1)$
10: $\phi' \coloneqq [\,\phi' \mid i = e \mapsto t\,]$
11: **return** $\Gamma \mid \Psi \vdash \mathsf{fill}^{\mathbf{0}\to\mathbf{1}}\ k.[\,\phi' - (k = \mathbf{0})\,]\ (\phi'[k = \mathbf{0}]) : [\,\phi\,]$

Given a goal boundary ϕ, we search for solutions either by natural fillers or by higher-dimensional fillers constructed with KanCube on line 4. In KanCube, we first select a set of sides that are left open on line 6 and then pick a solution to the corresponding KanCSP on line 7, which will fill all sides not left open with contorted cells. Finally, we call KanSolver recursively on the open sides on line 9, where $[\,\phi'[i = e]\,]$ denotes the boundary at $i = e$ induced by the faces already present in ϕ'.

The choices of solutions and open sides on lines 4, 6, 7 and 9 are made non-deterministically, which is done using the list monad in the solver discussed in §5. In practice, the performance of the algorithm depends heavily on the choices we make at this point. In our implementation, we first try to solve KanCSP with $Ope = \emptyset$. If contortions are not enough to construct all sides, it is useful to first use natural fillers which are induced by the goal boundary. In addition, it is expedient to incrementally increase the number of open sides solutions of KanCSP, e.g., using the depth-parameter d.

We now embed Algorithms 1 and 2 into a complete search procedure for Kan in Algorithm 3. Our Solver starts by trying to contort some cell of the cell context into the goal boundary. If this fails, we perform iterative deepening on the level of nested Kan cells.

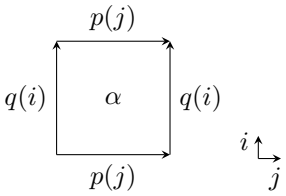
Algorithm 3 A solver for boundary problems.

Input: $\Gamma \mid \Psi \vdash \phi$ bdy
Output: $\Gamma \mid \Psi \vdash t : [\,\phi\,]$, if $\textsc{Kan}(\Gamma, \Psi, \phi)$ is solvable

1: **procedure** $\textsc{Solver}(\Gamma, \Psi, \phi)$
2: **for** $p \in \Gamma$ **do**
3: $t \leftarrow \textsc{Contort}(\Gamma, \Psi, \phi, p)$
4: **if** $t \neq$ `Unsolvable` **then**
5: **return** t
6: **for** $d \in \{1, \ldots\}$ **do**
7: $t \leftarrow \textsc{KanSolver}(\Gamma, \Psi, \phi, d)$
8: **if** $t \neq$ `Unsolvable` **then**
9: **return** t

▶ **Example 21** (Sq→Comp). To complete the proof of Eckmann-Hilton, we need to fill the cube from Figure 1(a) using Figure 1(b). We can do this already at a lower dimension – which means at greater generality: the cube from Figure 1(b) is captured with a square

$$\Gamma := \begin{cases} x : [\,], \\ p(i) : [\, i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto x \,], \\ q(i) : [\, i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto x \,], \\ \alpha(i,j) : \begin{bmatrix} i = \mathbf{0} \mapsto p(j) & \mid j = \mathbf{0} \mapsto q(i) \\ i = \mathbf{1} \mapsto p(j) & \mid j = \mathbf{1} \mapsto q(i) \end{bmatrix} \end{cases}$$



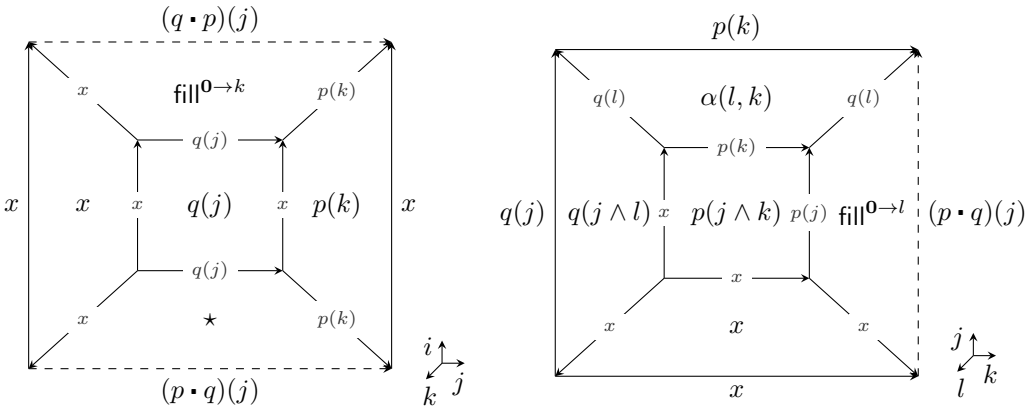
that we want to turn into a square with both path concatenations on opposite sides:

$$\Gamma \mid i,j \vdash ? : [\, i = \mathbf{0} \mapsto (p \cdot q)(j) \mid i = \mathbf{1} \mapsto (q \cdot p)(j) \mid j = \mathbf{0} \mapsto x \mid j = \mathbf{1} \mapsto x \,] \tag{5}$$

Incidentally, Γ is the list of generators of the HIT capturing the Torus in `agda/cubical`, while boundary (5) captures T^2, the definition of the torus in the HoTT book [35]. A solution to this problem thus induces a map from the cubical torus to the HoTT book torus.

We solve (5) using Algorithm 3. After seeing that we cannot solve this goal with a contortion, the algorithm at some point reaches depth $d = 3$ and solves KanCSP with open sides $Ope = \{(i = \mathbf{0}), (i = \mathbf{1})\}$. A solution to this CSP has the constant x square for $j = \mathbf{0}$, $p(k)$ for $j = \mathbf{1}$ and $q(j)$ for $k = \mathbf{0}$ as depicted in the left cube below.

When calling KanSolver recursively on the two missing sides, we find with KanFill that the $i = \mathbf{1}$ side can be solved with the natural filler for $q \cdot p$. To fill side $i = \mathbf{0}$, we again have to construct an open cube. One solution of KanCSP for this open cube is depicted on the right below. The $k = \mathbf{1}$ side is filled by the natural filler for $p \cdot q$. The other sides can be filled with contortions, where side $j = \mathbf{1}$ makes use of α.



5 A practical solver for Cubical Agda boundary problems

We have implemented the solver in `Haskell`,[4] providing the first experimental solver for boundary problems coming from `Cubical Agda`. The implementation of KanCSP is based on a monadic solver for finite domain constraint satisfaction problems [25]. The user inputs problems in a `.cube` file which contains a cell context and boundary problems over that context. If the solver finds a solution, it is printed in `Cubical Agda` syntax so that it can be copied and pasted into proof goals. Proper integration into `Cubical Agda` that allows the solver to be called as a tactic from `Agda` is work in progress.

[4] Source code and examples are available at: `https://github.com/maxdore/dedekind`

We have curated a small benchmarking suite of boundary problems, many of which are from the `agda/cubical` library. The problems are common proof obligations, such as associativity of path concatenation, rearrangements of sides of cubes, etc. On a standard laptop, all problems are quickly solved (often in < 50ms). This means that the solver is fast enough to fit seamlessly into a formalisation workflow and can be used as a tactic for solving routine proof goals. It can also solve some more complex goals such as Example 20.

In `Cubical Agda`, the constant path at x of type $x \equiv x$ is expressed with λ-abstraction as $\lambda\ i \to x$. We can use the PathP type to describe higher-dimensional boundaries, e.g., PathP $(\lambda\ j \to x \equiv x)\ (\lambda\ i \to x)\ (\lambda\ i \to x)$ is the boundary of a square with reflexive paths on its sides. Given two such squares p and q, The Eckmann-Hilton cube is derived in ∼150ms:

```
EckmannHilton-Cube : PathP (λ i → q i ≡ q i) p p
EckmannHilton-Cube = λ i j k → hcomp (λ l → λ {
  (i = i0) → p j (k ∧ l) ; (j = i0) → q i k ; (k = i0) → x ;
  (i = i1) → p j (k ∧ l) ; (j = i1) → q i k ; (k = i1) → p j l }) (q i k)
```

The `Cubical Agda` primitive hcomp captures Kan fillers in direction $\mathbf{0} \to \mathbf{1}$. The solution to the boundary problem discussed in the Sq→Comp example is found in ∼15ms, its translation into `Cubical Agda` looks as follows (manually compressed to not use too much space in the paper; the actual pretty-printed output is more readable):

```
Sq→Comp : PathP (λ j → q j ≡ q j) p p → p ∙ q ≡ q ∙ p
Sq→Comp α i j = hcomp (λ k → λ {
  (i = i0) → hcomp (λ l → λ {
    (j = i0) → x ; (k = i0) → q (j ∧ l) ; (j = i1) → α l k ;
    (k = i1) → hfill (λ m → λ { (j = i0) → x ; (j = i1) → q m }) (inS (p j)) l })
    (p (j ∧ k)) ;
  (i = i1) → hfill (λ l → λ { (j = i0) → x ; (j = i1) → p l }) (inS (q j)) k ;
  (j = i0) → x ; (j = i1) → p k })
  (q j)
```

The function hfill $\phi\ t\ i$ is used in `agda/cubical` to define fillers in direction $\mathbf{0} \to i$. The term t has to be embedded into the cube structure using inS, which is inserted automatically by the `Cubical Agda` syntax pretty-printer of the solver.

Using these two automatically constructed proofs, we can readily establish by hand the classical formulation of the Eckmann-Hilton argument in terms of path concatenations:

```
EckmannHilton : p ∙ q ≡ q ∙ p
EckmannHilton = Sq→Comp p q EckmannHilton-Cube
```

The boundary problem posed by EckmannHilton can also be passed directly to our solver, however, it is not yet able to prove this problem within 100s. We have also curated some further boundary problems which cannot be solved at the moment, these include a 7-dimensional analogue of the Square to cube contortion example and the syllepsis [28], which establishes a higher coherence property of the Eckmann-Hilton proof.

In summary, while there is room to make the solver more performant, it can quickly prove technical lemmas for us that would be tedious to prove by hand, taking significant proof burden from a user of `Cubical Agda`. Furthermore, some deeper results of synthetic homotopy theory, like the Eckmann-Hilton argument, can also be proved if the statement is phrased carefully.

6 Future and related work

There are many ways in which our work can be extended: the performance of the solver can be improved by exploring other heuristics and refinements of the algorithms; the solver should be properly integrated into `Cubical Agda`; the contortion theory should be extended to also include the De Morgan involution. The solver could be extended to problems involving multiple types and functions and to use cubical type theory's *transport* primitive.

Early work on proof automation in HoTT is Brunerie's work on computer-generated proofs for the monoidal structure of smash products [7] which used path-induction and metaprogramming in `Agda`. The problem of deciding equality in the cofibration logic of cubical type theories has been studied by [27]. Among other things, they also establish complexity-related results, in particular, that the entailment problems of the cofibration languages of [1] and [9] are coNP-complete. Another line of related work where the relationship to our work needs to be better understood is higher-dimensional rewriting, in particular, those based on ∞-categories [14], operads [29], polygraphs [3] and associative n-categories [11].

References

1 Carlo Angiuli, Guillaume Brunerie, Thierry Coquand, Robert Harper, Kuen-Bang Hou (Favonia), and Daniel R. Licata. Syntax and models of Cartesian cubical type theory. *Mathematical Structures in Computer Science*, 31(4):424–468, 2021. doi:10.1017/S0960129521000347.

2 Carlo Angiuli, Kuen-Bang Hou (Favonia), and Robert Harper. Cartesian cubical computational type theory: Constructive reasoning with paths and equalities. In *27th EACSL Annual Conference on Computer Science Logic, CSL 2018, September 4-7, 2018, Birmingham, UK*, pages 6:1–6:17, 2018. doi:10.4230/LIPIcs.CSL.2018.6.

3 Dimitri Ara, Albert Burroni, Yves Guiraud, Philippe Malbos, François Métayer, and Samuel Mimram. Polygraphs: From rewriting to higher categories. Submitted, 2023. arXiv:2312.00429.

4 Steve Awodey. Cartesian cubical model categories, 2023. arXiv:2305.00893.

5 Ronald Brown and Philip J. Higgins. On the algebra of cubes. *Journal of Pure and Applied Algebra*, 21(3):233–260, 1981. doi:10.1016/0022-4049(81)90018-9.

6 Guillaume Brunerie. *On the homotopy groups of spheres in homotopy type theory*. PhD thesis, Université de Nice Sophia Antipolis, 2016.

7 Guillaume Brunerie. Computer-generated proofs for the monoidal structure of the smash product. *Homotopy Type Theory Electronic Seminar Talks*, November 2018. URL: https://www.uwo.ca/math/faculty/kapulkin/seminars/hottest.html.

8 J. Daniel Christensen and Luis Scoccola. The Hurewicz theorem in Homotopy Type Theory. *Algebraic & Geometric Topology*, 23:2107–2140, 2023.

9 Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom. In Tarmo Uustalu, editor, *21st International Conference on Types for Proofs and Programs (TYPES 2015)*, volume 69 of *LIPIcs*, pages 5:1–5:34. Schloss Dagstuhl, 2018. doi:10.4230/LIPIcs.TYPES.2015.5.

10 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.

11 Christoph Dorn. *Associative n-categories*. PhD thesis, University of Oxford, 2018.

12 Beno Eckmann and Peter J. Hilton. Group-like structures in general categories I: multiplications and comultiplications. *Mathematische Annalen*, 145:227–255, 1962.

13 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)*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005124.

14 Eric Finster, David Reutter, Jamie Vicary, and Alex Rice. A type theory for strictly unital ∞-categories. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12, 2022. doi:10.1145/3531130.3533363.

15 Lennart Van Hirtum, Patrick De Causmaecker, Jens Goemaere, Tobias Kenter, Heinrich Riebler, Michael Lass, and Christian Plessl. A computation of D(9) using FPGA Supercomputing, 2023. arXiv:2304.03039.

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

17 Kuen-Bang Hou (Favonia) and Michael Shulman. The Seifert-van Kampen Theorem in Homotopy Type Theory. In Jean-Marc Talbot and Laurent Regnier, editors, *25th EACSL Annual Conference on Computer Science Logic (CSL 2016)*, volume 62 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 22:1–22:16, Dagstuhl, Germany, 2016. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CSL.2016.22.

18 Peter Tennant Johnstone. *Stone Spaces*, volume 3 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, 1986.

19 Christian Jäkel. A computation of the ninth Dedekind Number, 2023. arXiv:2304.00895.

20 Daniel M. Kan. Abstract homotopy. I. *Proceedings of the National Academy of Sciences of the United States of America*, 41(12):1092–1096, 1955.

21 Jean-Louis Laurière. A language and a program for stating and solving combinatorial problems. *Artificial Intelligence*, 10(1):29–127, 1978. doi:10.1016/0004-3702(78)90029-2.

22 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.

23 Anders Mörtberg and Loïc Pujet. Cubical Synthetic Homotopy Theory. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2020, pages 158–171, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3372885.3373825.

24 Ian Orton and Andrew M. Pitts. Axioms for Modelling Cubical Type Theory in a Topos. *Logical Methods in Computer Science*, Volume 14, Issue 4, December 2018. doi:10.23638/LMCS-14(4:23)2018.

25 David Overton. Constraint Programming in Haskell. Melbourne Haskell Users Group, 2015. URL: https://de.slideshare.net/davidoverton/constraint-programming-in-haskell.

26 Emily Riehl and Michael Shulman. A type theory for synthetic ∞-categories. *Higher Structures*, 1(1):116–193, 2017.

27 Matthew Z. Weaver Robert Rose and Dan R. Licata. Deciding entailment for cofibration languages. At the second international conference on *Homotopy Type Theory*, 2023. URL: https://hott.github.io/HoTT-2023/slides/rose.pdf.

28 Kristina Sojakova and G. A. Kavvos. Syllepsis in Homotopy Type Theory. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. Association for Computing Machinery, 2022. doi:10.1145/3531130.3533347.

29 Cédric Ho Thanh, Pierre-Louis Curien, and Samuel Mimram. A sequent calculus for opetopes. In *Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12, 2019. doi:10.1109/LICS.2019.8785667.

30 The 1Lab Development Team. The 1Lab, 2023. URL: https://1lab.dev.

31 The Agda Community. Cubical Agda Library, October 2023. URL: https://github.com/agda/cubical.

32 The RedPRL Development Team. The `redtt` proof assistant, 2018. URL: https://github.com/RedPRL/redtt/.

33 The RedPRL Development Team. The `cooltt` proof assistant, 2021. URL: https://github.com/RedPRL/cooltt/.

34 Edward Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993. doi:10.1016/C2013-0-07627-X.

35 The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study, 2013.

36 Floris van Doorn. *On the Formalization of Higher Inductive Types and Synthetic Homotopy Theory*. PhD thesis, Carnegie Mellon University, May 2018. URL: https://arxiv.org/abs/1808.10690.

37 Andrea Vezzosi, Anders Mörtberg, and Andreas Abel. Cubical Agda: a Dependently Typed Programming Language With Univalence and Higher Inductive Types. *Proceedings of the ACM on Programming Languages*, 3(ICFP):1–29, 2019. doi:10.1145/3341691.

38 Vladimir Voevodsky. The equivalence axiom and univalent models of type theory, February 2010. Notes from a talk at Carnegie Mellon University, available at http://www.math.ias.edu/vladimir/files/CMU_talk.pdf.

Mirroring Call-By-Need, or Values Acting Silly

Beniamino Accattoli
Inria & LIX, Ecole Polytechnique, UMR 7161, France

Adrienne Lancelot
Inria & LIX, Ecole Polytechnique, UMR 7161, France
Université Paris Cité, CNRS, IRIF, Paris, France

Abstract

Call-by-need evaluation for the λ-calculus can be seen as merging the best of call-by-name and call-by-value, namely the wise erasing behaviour of the former and the wise duplicating behaviour of the latter. To better understand how duplication and erasure can be combined, we design a degenerated calculus, dubbed *call-by-silly*, that is symmetric to call-by-need in that it merges the worst of call-by-name and call-by-value, namely silly duplications by-name and silly erasures by-value.

We validate the design of the call-by-silly calculus via rewriting properties and multi types. In particular, we mirror the main theorem about call-by-need – that is, its operational equivalence with call-by-name – showing that call-by-silly and call-by-value induce the same contextual equivalence. This fact shows the blindness with respect to efficiency of call-by-value contextual equivalence. We also define a call-by-silly *strategy* and measure its length via tight multi types. Lastly, we prove that the call-by-silly strategy computes evaluation sequences of maximal length in the calculus.

2012 ACM Subject Classification Theory of computation → Lambda calculus

Keywords and phrases Lambda calculus, intersection types, call-by-value, call-by-need

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.23

Related Version *Full Version*: `https://arxiv.org/abs/2402.12078` [14]

1 Introduction

Plotkin's call-by-value and Wadsworth's call-by-need (untyped) λ-calculi were introduced in the 1970s as more application-oriented variants of the ordinary call-by-name λ-calculus [53, 49]. The simpler call-by-value (shortened to CbV) calculus has found a logical foundation in formalisms related to classical logic or linear logic [28, 44, 45, 33].

The foundation of call-by-need (CbNeed) is less developed, particularly its logical interpretation. The duality related to classical logic can accommodate CbNeed, as shown by Ariola et al. [19, 17], but it does not provide an explanation for it. Within linear logic, CbNeed is understood as a sort of *affine* CbV, according to Maraist et al. [47]. Such an interpretation however is unusual, because it does not match exactly with cut-elimination in linear logic, as for call-by-name (CbN) and CbV, and it is rather connected with *affine logic*.

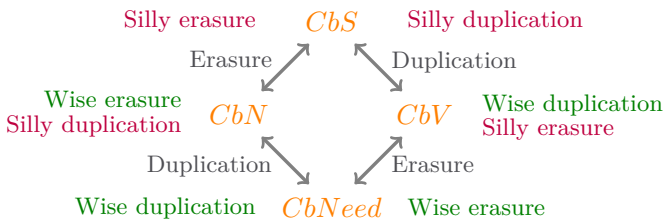
CbNeed Optimizes CbN. The main foundational theorem for CbNeed is the *operational equivalence of CbN and CbNeed* due to Ariola et al. [18], that is, the fact that they induce the same contextual equivalence, despite being based on different evaluation mechanisms. The result formalizes that CbNeed is a semantic-preserving optimization of CbN.

An elegant semantic proof of this result is given by Kesner [38]. She shows that the CbN multi type system by de Carvalho [30, 31] (considered before his seminal work also by Gardner [35] and Kfoury [43]), which characterizes termination for CbN (that is, t is typable $\Leftrightarrow$ t is CbN terminating), characterizes termination also for CbNeed. Multi types are also known as *non-idempotent intersection types*, and have strong ties with linear logic.

Duplication and Erasure, Wise and Silly. The linear logic interpretation of CbN and CbV provides some operational insights about CbNeed. By bringing to the fore duplication and erasure, linear logic underlines a second symmetry between CbN and CbV, which is independent of classical logic as it can already be observed in an intuitionistic setting. The idea is that CbN is *wise* with respect to erasure, because it never evaluates arguments that might be erased (for instance $(\lambda x.\mathtt{I})\Omega \to_{CbN} \mathtt{I}$, where $\mathtt{I}$ is the identity and Ω is the looping combinator), while it is *silly* with respect to duplications, as it repeats possibly many times the evaluation of arguments that are used at least once (for instance, $(\lambda x.xx)(\mathtt{II}) \to_{CbN} \mathtt{II}(\mathtt{II})$). Symmetrically, CbV is silly with respect to erasures, as it reduces in arguments that are not going to be used (for instance looping on $(\lambda x.\mathtt{I})\Omega$), but it is wise for duplications, as it reduces only once arguments that shall be used at least once (e.g. $(\lambda x.xx)(\mathtt{II}) \to_{CbV} (\lambda x.xx)\mathtt{I}$).

CbNeed is Pure Wiseness. In this framework, one can see CbNeed as merging the best of CbN and CbV, i.e. wise erasure and wise duplication. In [12], this insight guides Accattoli et al. in extending Kesner's above-mentioned study of CbNeed via multi types [38]. Starting from existing multi type systems for CbN and CbV, they pick out the aspects for CbN wise erasures and CbV wise duplication as to build a CbNeed multi type system. Their system characterizes CbNeed termination, as does the CbN system in [38]. Additionally, it tightly characterizes CbNeed evaluation *quantitatively*: from type derivations one can read the *exact* number of CbNeed evaluation steps, which is not possible in the CbN system.

Pure Silliness. This paper studies an unusual – and at first counter-intuitive – combination of duplication and erasure. The idea is to mix together silly erasure and silly duplication, completing the following diagram of strategies with its new *call-by-silly* (CbS) corner:



(1)

Designing the CbS calculus and a CbS evaluation strategy is of no interest for programming purposes, as CbS is desperately inefficient, by construction. It is theoretically relevant, however, because it showcases how modularly duplication and erasure can be combined. As we shall discuss, the study of CbS also contributes to the understanding of CbV.

Quantitative Goal and Micro Steps. We introduce CbS and provide evidence of its good design via contributions that mirror as much as possible the theory of CbNeed. Our ultimate test is to mirror the quantitative study of CbNeed by Accattoli et al. [12], via a new system of multi types for CbS. It is a challenging design goal because this kind of quantitative results requires a *perfect matching* between the operational and the multi types semantics.

As we explain at the end of Sect. 9, such a goal does not seem to be cleanly achievable within the λ-calculus, where duplication is *small-step*, that is, arguments are substituted on *all* the occurrences of a variable at once. It is instead attainable in a *micro-step* setting with explicit substitutions, where one copy at a time is performed. Moreover, in the λ-calculus duplication and erasure are hard to disentangle as they are both handled by β-reduction.

Our Framework. A good setting for our purpose is Accattoli and Kesner's *linear substitution calculus* (LSC) [1, 8], a calculus with explicit substitutions which is a variation over a calculus by Milner [48, 39] exploiting ideas from another work of theirs [13]. A key feature of the LSC is that it has *separate* rules for duplication and erasure, and *just one rule for each*.

The LSC comes in two dialects, CbN and CbV, designed over linear logic proof nets [3, 2]. The further CbNeed LSC is obtained by taking the duplication rule of the CbV LSC and the erasing rule of the CbN LSC. It was first defined by Accattoli et al. [5] and then studied or extended by many recent works on CbNeed [38, 21, 6, 7, 41, 23, 12, 40, 22, 15].

Our starting point is the principled definition of the new *silly (linear) substitution calculus* (SSC), the variant of the LSC obtained by mirroring the construction for CbNeed, namely by putting together the CbN rule for duplication and the CbV one for erasure. In this way, all the corners of the strategy diagram (1) fit into the same framework. Since CbNeed is usually studied with respect to weak evaluation (that is, not under abstraction), we only consider weak evaluation for the SSC (and leave the strong, unrestricted case to future work).

Contribution 1: Rewriting Properties. After defining the SSC, we prove some of the rewriting properties that are expected from every well-behaved calculus. Namely, we provide a characterization of its normal forms, that reduction is confluent, and that – as it is the case for all dialects of the LSC – the CbV erasing rule can be postponed.

Via the multi types of the next contribution, we also prove *uniform normalization* for the SSC, that is, the fact that a term is weakly normalizing (i.e. it reduces to a normal form) if and only if it is strongly normalizing (i.e. it has no diverging reductions). This is trivially true in (weak) CbV, where diverging sub-terms cannot be erased and terms with redexes cannot be duplicated, and false in CbN and CbNeed, where diverging sub-terms can be erased. In the SSC, it is true but non-trivial, because terms with redexes can be duplicated.

Contribution 2: Operational Equivalence. Next, we develop the mirrored image of the main qualitative result for CbNeed, that is, we prove the operational equivalence of CbS and CbV. We do so by mirroring Kesner's semantic proof via multi types [38].

To this purpose, we cannot use the existing multi type system for CbV, due to Ehrhard [32], as Kesner's proof is based on a system for the *slower* of the two systems, in her case CbN. Therefore, we introduce a silly multi type system, designed in a dual way to the one for CbNeed by Accattoli et al. [12].

Our system is a minor variant over a system for CbN strong normalization by Kesner and Ventura [42], and it also bears strong similarities with a system for CbV by Manzonetto et al. [46, 37]. We prove that it characterizes termination for closed terms in both CbV and CbS, from which it follows the coincidence of their respective contextual equivalences. This result shows that CbV is a semantic-preserving optimization of CbS, exactly as CbNeed is for CbN.

CbV Contextual Equivalence is Blind to Efficiency. Contextual equivalence is usually considered as *the* notion of program equivalence. This paper points out a fundamental limitation of contextual equivalence for the effect-free untyped CbV λ-calculus. In absence of effects, contextual equivalence does distinguish between CbV and CbN because of their different erasing policies (as they change the notion of termination) but it is unable to distinguish between their duplication policies (which only impact the efficiency of evaluation), because it is *blind to efficiency*: it cannot "count" how many times a sub-term is evaluated.

It is well known that, in richer frameworks with effects, CbV contextual equivalence can "count". In presence of state, indeed, more terms are discriminated, as doing once *or* twice the same operation can now change the content of a state and contexts are expressive enough to generate and distinguish these state changes [52, 25]. But it is instead not well known, or not fully digested, that in the pure setting this is not possible – thus that CbV contextual equivalence validates silly duplications – as we have repeatedly noted in discussions with surprised colleagues.

To the best of our knowledge, such a blindness to efficiency has not been properly established before. Proving contextual equivalence is technically difficult, often leading to hand-waving arguments. Moreover, figuring out a general characterization of silly duplications is tricky. We provide here an easy way out to prove CbV contextual equivalence for terms related by silly duplications: it is enough to reduce them to the same normal form in the SSC, a calculus qualitatively equivalent to CbV but based on silly duplications.

Note also that the equivalence of CbN and CbNeed is interpreted positively (despite implying that CbNeed contextual equivalence is blind to efficiency) because it gives a foundation to CbNeed. Curiously, the mirrored equivalence of CbV and CbS crystallizes instead a negative fact about pure CbV, since CbV is expected to be efficiency-sensitive.

Contribution 3: The CbS Strategy. Beyond the SSC, we specify a tricky notion of CbS *strategy*. CbS contexts are the modification of CbN contexts that also evaluate shared sub-terms when they are *no longer needed*, i.e., after all the needed copies have already been evaluated. The CbS strategy is an extension of the CbN strategy, on non-erasing steps.

We then prove that the CbS strategy is (essentially) deterministic (precisely, it is *diamond*, a weakened form of determinism), and that the CbV erasing rule of the strategy can be postponed. These usually simple results have in our case simple but lengthy proofs, because of the tricky grammar defining CbS contexts.

Contribution 4: Tight Types and (Maximal) Evaluation Lengths. Lastly, we show that the quantitative result for CbNeed by Accattoli et al. [12] is also mirrored by our silly type system. We mimic [12] and we extract the exact length of evaluations for the CbS strategy from a notion of *tight* type derivation.

We then use this quantitative result to show that the CbS strategy actually computes a *maximal* evaluation sequence in the weak SSC. This is not so surprising, given that maximal evaluations and strong normalization are related and measured with similar systems of multi types, as in Bernadet and Lengrand [24], Kesner and Ventura [42], and Accattoli et al. [9].

At the same time, however, the maximality of CbS is specific to weak evaluation and different from those results in the literature, which concern strong evaluation (that is, also under abstraction). In particular, CbS would *not* be maximal in a strong setting (which we do not treat here), as it would erase a value before evaluating it (as CbV also does). This is precisely the difference between our type system and that of Kesner and Ventura.

Related Work. The only work that might be vaguely reminiscent of ours is by Ariola et al. [19, 17], who study CbNeed with respect to the classical duality between CbN and CbV and control operators. Their work and ours however are orthogonal and incomparable: they do not obtain inefficient strategies and we do not deal with control operators.

Proofs. A long version with proofs in the Appendix is on ArXiv [14].

Language				Root rules			
Terms	t, s, u	$::=$	$x \mid \lambda x.t \mid ts \mid t[x{\leftarrow}s]$	$S\langle\lambda x.t\rangle s$	$\mapsto_{\mathtt{m}}$	$S\langle t[x{\leftarrow}s]\rangle$	
Values	v, v'	$::=$	$\lambda x.t$	$W\langle\!\langle x\rangle\!\rangle[x{\leftarrow}s]$	$\mapsto_{\mathtt{e}_W}$	$W\langle\!\langle s\rangle\!\rangle[x{\leftarrow}s]$	
Sub. Ctxs	S, S'	$::=$	$\langle\cdot\rangle \mid S[x{\leftarrow}s]$	$t[x{\leftarrow}S\langle v\rangle]$	$\mapsto_{\mathtt{gcv}}$	$S\langle t\rangle$	if $x \notin \mathtt{fv}(t)$

Weak reduction $\to_{\mathtt{w}}$

Weak contexts
$W ::= \langle\cdot\rangle \mid Wt \mid tW \mid t[x{\leftarrow}W] \mid W[x{\leftarrow}s]$

$$
\begin{array}{rcl}
\to_{\mathtt{wm}} & := & W\langle\mapsto_{\mathtt{m}}\rangle \\
\to_{\mathtt{we}} & := & W\langle\mapsto_{\mathtt{e}_W}\rangle \\
\to_{\mathtt{wgcv}} & := & W\langle\mapsto_{\mathtt{gcv}}\rangle \\
\to_{\mathtt{w}} & := & \to_{\mathtt{wm}} \cup \to_{\mathtt{we}} \cup \to_{\mathtt{wgcv}}
\end{array}
$$

Figure 1 The (weak) silly (linear) substitution calculus (SSC).

2 The Weak Silly Substitution Calculus

Inception. The SSC is a variant over Accattoli and Kesner's linear substitution calculus (LSC) [1, 8], which is a micro-step λ-calculus with explicit substitutions. *Micro-step* means that substitutions act on one variable occurrence at a time, rather than *small-step*, that is, on all occurrences at the same time. The LSC exists in two main variants, CbN and CbV. The CbV variant is usually presented with small-step rules, and called *value substitution calculus*, the micro-step (or linear) variant of which appears for instance in Accattoli et al. [5, 12]. The two calculi have similar and yet different duplication and erasing rewriting rules. We refer to [5, 12] for a uniform presentation of CbN, CbV, and CbNeed in the LSC.

The SSC in Fig. 1 is obtained dually to CbNeed, taking the duplication rule $\mapsto_{\mathtt{e}_W}$ of the CbN LSC and the erasing rule $\mapsto_{\mathtt{gcv}}$ of the CbV LSC. Its evaluation is micro-step. We only define its weak evaluation, which does not enter into abstractions, as it is often done in comparative studies between CbN, CbV, and CbNeed such as [5, 12]. This sections overviews the definitions in the figure. The SSC deals with possibly open terms. The CbS strategy, to be defined in Sect. 8, shall instead deal only with closed terms.

Terms. Terms of the SSC are the same as for the LSC and extend the λ-calculus with *explicit substitutions* $t[x{\leftarrow}s]$ (shortened to ESs), that is a more compact notation for $\mathsf{let}\ x = s\ \mathsf{in}\ t$, but where the order of evaluation between t and s is a priori not fixed. The set $\mathtt{fv}(t)$ of *free* variables of a term t is defined as expected, in particular, $\mathtt{fv}(t[x{\leftarrow}s]) := (\mathtt{fv}(t) \setminus \{x\}) \cup \mathtt{fv}(s)$. Both $\lambda x.t$ and $t[x{\leftarrow}s]$ bind x in t, and terms are considered up to α-renaming. A term t is *closed* if $\mathtt{fv}(t) = \emptyset$, *open* otherwise. As usual, terms are identified up to α-equivalence. Meta-level capture-avoiding substitution is noted $t\{x{\leftarrow}s\}$. Note that values are only abstraction; this choice shall be motivated in the next section.

Contexts. Contexts are terms with exactly one occurrence of the *hole* $\langle\cdot\rangle$, an additional constant, standing for a removed sub-term. We shall use many different contexts. The most general ones in this paper are *weak contexts* W, which simply allow the hole to be anywhere but under abstraction. To define the rewriting rules, *substitution contexts* S (i.e. lists of explicit substitutions) also play a role. The main operation about contexts is *plugging* $W\langle t\rangle$ where the hole $\langle\cdot\rangle$ in context W is replaced by t. Plugging, as usual with contexts, can capture variables – for instance $((\langle\cdot\rangle t)[x{\leftarrow}s])\langle x\rangle = (xt)[x{\leftarrow}s]$. We write $W\langle\!\langle t\rangle\!\rangle$ when we want to stress that the context W does not capture the free variables of t.

Rewriting Rules. The reduction rules of the SSC are slightly unusual as they use *contexts* both to allow one to reduce redexes located in sub-terms, which is standard, *and* to define the redexes themselves, which is less standard. This approach is called *at a distance* and related to cut-elimination on proof nets (from which the terminology *multiplicative* and *exponential* to be discussed next is also taken), see Accattoli [2, 3].

The *multiplicative rule* $\mapsto_{\mathtt{m}}$ is essentially the β-rule, except that the argument goes into a new ES, rather then being immediately substituted, and that there can be a substitution context S in between the abstraction and the argument. Example: $(\lambda x.y)[y{\leftarrow}t]s \mapsto_{\mathtt{m}} y[x{\leftarrow}s][y{\leftarrow}t]$. One with on-the-fly α-renaming is $(\lambda x.y)[y{\leftarrow}t]y \mapsto_{\mathtt{m}} z[x{\leftarrow}y][z{\leftarrow}t]$.

The *exponential rule* $\mapsto_{\mathtt{e}_W}$ replaces a single variable occurrence, the one appearing in the context W. Example: $(xx)[x{\leftarrow}y[y{\leftarrow}t]] \mapsto_{\mathtt{e}_W} (x(y[y{\leftarrow}t]))[x{\leftarrow}y[y{\leftarrow}t]]$. The notation $\mapsto_{\mathtt{e}_W}$ stresses that the rule is parametric in a notion of context W, that specifies where the variable replacements are allowed, and which shall be exploited to define the CbS strategy in Sect. 8.

The *garbage collection (GC) rule by value* $\mapsto_{\mathtt{gcv}}$ eliminates a value v, keeping the list of substitutions S previously surrounding the value, which might contain terms that are not values, and so cannot be erased with v. This rule is the CbV erasing rule. Example: $(\lambda z.yy)[x{\leftarrow}\mathtt{I}[w{\leftarrow}t]] \mapsto_{\mathtt{gcv}} (\lambda z.yy)[w{\leftarrow}t]$, where $\mathtt{I}$ is the identity.

The three root rules $\mapsto_{\mathtt{m}}$, $\mapsto_{\mathtt{e}_W}$, and $\mapsto_{\mathtt{gcv}}$ are then closed by weak contexts. We shorten $W\langle t\rangle \to_{\mathtt{wm}} W\langle s\rangle$ if $t \mapsto_{\mathtt{m}} s$ with $\to_{\mathtt{wm}} := W\langle \mapsto_{\mathtt{m}}\rangle$, and similarly for the other rules. The reduction $\to_{\mathtt{w}}$ encompasses all possible reductions in the weak SSC.

Silliness Check and the Silly Extra Copy. Let us evaluate with the SSC the examples used in the introduction to explain silly and wise behaviour. About silly erasures, $(\lambda x.\mathtt{I})\Omega$ diverges, as in CbV. One can reduce it to $\mathtt{I}[x{\leftarrow}\Omega]$, but then there is no way of erasing Ω, which is not a value and does not reduce to a value. About silly duplications, $(\lambda x.xx)(\mathtt{II})$ can reduce both to $\mathtt{II}(\mathtt{II})$ and to $(\lambda x.xx)\mathtt{I}$, as in CbN. The silly aspect is the fact that one *can* reduce to $\mathtt{II}(\mathtt{II})$, since in CbV this is *not* possible. The CbS strategy of Sect. 8 shall always select the silly option.

Note that $(\lambda x.xx)(\mathtt{II})$ reduces to $(xx)[x{\leftarrow}\mathtt{II}]$ and then *three* copies of $\mathtt{II}$ can be evaluated, as one can reduce to $\mathtt{II}(\mathtt{II})[x{\leftarrow}\mathtt{II}]$ and the copy in the ES has to be evaluated before being erased, because it is not a value. Such a further copy would not be evaluated in CbN, as terms are erased without being evaluated. We refer to this aspect as to the *silly extra copy*.

3 Basic Rewriting Notions

Given a rewriting relation $\to_{\mathsf{r}}$, we write $d\colon t \to_{\mathsf{r}}^* s$ for a $\to_{\mathsf{r}}$-reduction sequence from t to s, the length of which is noted $|d|$. Moreover, we use $|d|_a$ for the number of a-*steps* in d, for a sub-relation $\to_a$ of $\to_{\mathsf{r}}$.

A term t is *weakly* r-*normalizing*, noted $t \in \mathrm{WN}_{\mathsf{r}}$, if $d\colon t \to_{\mathsf{r}}^* s$ with s r-normal; and t is *strongly* r-*normalizing*, noted $t \in \mathrm{SN}_{\mathsf{r}}$, if there are no diverging r-sequences from t, or, equivalently, if all its reducts are in SN_{r}.

According to Dal Lago and Martini [29], a relation $\to_{\mathsf{r}}$ is *diamond* if $s_1 \,{}_{\mathsf{r}}{\leftarrow} t \to_{\mathsf{r}} s_2$ and $s_1 \neq s_2$ imply $s_1 \to_{\mathsf{r}} u \,{}_{\mathsf{r}}{\leftarrow} s_2$ for some u. If $\to_{\mathsf{r}}$ is diamond then:

1. *Confluence*: $\to_{\mathsf{r}}$ is confluent, that is, $s_1 \,{}_{\mathsf{r}}^*{\leftarrow} t \to_{\mathsf{r}}^* s_2$ implies $s_1 \to_{\mathsf{r}}^* u \,{}_{\mathsf{r}}^*{\leftarrow} s_2$ for some u;
2. *Length invariance*: all r-evaluations to normal form with the same start term have the same length (i.e. if $d\colon t \to_{\mathsf{r}}^* s$ and $d'\colon t \to_{\mathsf{r}}^* s$ with s $\to_{\mathsf{r}}$-normal then $|d| = |d'|$);
3. *Uniform normalization*: t is weakly r-normalizing if and only if it is strongly r-normalizing.

Basically, the diamond captures a more liberal form of determinism.

4 Rewriting Properties

In this section, we study some rewriting properties of the weak SSC. We give a characterization of weak normal forms and prove postponement of garbage collection by value and confluence.

Characterization of Normal Forms. The characterization of (possibly open) weak normal forms requires the following concept.

▶ **Definition 1** (Shallow free variables). *The set* $\mathtt{shfv}(t)$ *of* shallow free variables *of* t *is the set of variables with occurrences out of abstractions in* t*:*

$$\begin{array}{lcllcl} \mathtt{shfv}(x) & := & \{x\} & \mathtt{shfv}(\lambda x.t) & := & \emptyset \\ \mathtt{shfv}(ts) & := & \mathtt{shfv}(t) \cup \mathtt{shfv}(s) & \mathtt{shfv}(t[x{\leftarrow}s]) & := & (\mathtt{shfv}(t) \setminus \{x\}) \cup \mathtt{shfv}(s) \end{array}$$

▶ **Proposition 2.** *t is $\to_{\mathtt{w}}$-normal if and only if t is a weak normal term according to the following grammar:*

$$\begin{array}{rrcll} \textsc{Weak answers} & a, a & ::= & v & \mid a[x{\leftarrow}i] \text{ with } x \notin \mathtt{shfv}(a) \\ & & & & \mid a[x{\leftarrow}a'] \text{ with } x \in \mathtt{fv}(a) \setminus \mathtt{shfv}(a) \\ \textsc{Inert terms} & i, i' & ::= & x \mid in & \mid i[x{\leftarrow}i'] \text{ with } x \notin \mathtt{shfv}(i) \\ & & & & \mid i[x{\leftarrow}a] \text{ with } x \in \mathtt{fv}(i) \setminus \mathtt{shfv}(i) \\ \textsc{Weak normal terms} & n, n' & ::= & a \mid i & \end{array}$$

Postponement of GC by Value. As it is usually the case in all the dialects of the LSC, the erasing rule of the weak SSC, that is $\to_{\mathtt{wgcv}}$, can be postponed. Let $\to_{\mathtt{w}\neg\mathtt{gcv}} := \to_{\mathtt{wm}} \cup \to_{\mathtt{we}}$.

▶ **Proposition 3** (Postponement of garbage collection by value). *If $d : t \to_{\mathtt{w}}^{*} s$ then $t \to_{\mathtt{w}\neg\mathtt{gcv}}^{k} \to_{\mathtt{wgcv}}^{h} s$ with $k = |d|_{\mathtt{w}\neg\mathtt{gcv}}$ and $h \geq |d|_{\mathtt{wgcv}}$.*

Local Termination. To prepare for confluence, we recall the following crucial property, which is essentially inherited from the LSC, given that termination for $\to_{\mathtt{wgcv}}$ is trivial.

▶ **Proposition 4** (Local termination). *Reductions $\to_{\mathtt{wm}}$, $\to_{\mathtt{we}}$, and $\to_{\mathtt{wgcv}}$ are strongly normalizing separately.*

Confluence. The proof of confluence for the weak SSC given here is a minor variant of what would be done for the LSC, except that there is no direct proof of confluence in the literature for the LSC[1]. Overviewing the proof allows us to explain a design choice of the SSC.

The proof is based on an elegant technique resting on local diagrams and local termination, namely the Hindley-Rosen method. In our case, it amounts to prove that the three rules $\to_{\mathtt{wm}}$, $\to_{\mathtt{we}}$, and $\to_{\mathtt{wgcv}}$ are confluent separately, proved by local termination and Newman's lemma, and commute, proved by local termination and Hindley's strong commutation. Confluence then follows by Hindley-Rosen lemma, for which the union of confluent and commuting reductions is confluent. The Hindley-Rosen method is a modular technique often used for confluence of extensions of the λ-calculus, for instance in [20, 34, 51, 27, 50, 26, 18, 16, 4].

▶ **Lemma 5** (Local confluence). *Reductions $\to_{\mathtt{wm}}$ and $\to_{\mathtt{wgcv}}$ are diamond, $\to_{\mathtt{we}}$ is locally confluent.*

[1] Confluence of the LSC holds, as it follows from stronger results about residuals in Accattoli et al. [8] but here we want to avoid the heaviness of residuals.

In contrast to confluence, commutation of two reductions $\to_1$ and $\to_2$ does *not* follow from their *local* commutation and strong normalization. In our case, however, the rules verify a non-erasing form of Hindley's strong (local) commutation [36] of $\to_1$ over $\to_2$, here dubbed *strict strong commutation*: if $s_1\ {}_1\!\leftarrow t \to_2 s_2$ then $\exists u$ such that $s_1 \to_2^+ u\ {}_1\!\leftarrow s_2$, that is, on one side of the commutation there are no duplications of steps, and on both sides there are no erasures, because $\to_{\mathtt{wgcv}}$ can only erase values, which do not contain redexes since evaluation is weak. Strong commutation and strong normalization do imply commutation.

▶ **Lemma 6** (Strict strong local commutations). *Reduction $\to_{\mathtt{we}}$ (resp. $\to_{\mathtt{we}}$; resp. $\to_{\mathtt{wgcv}}$) strictly strongly commutes over $\to_{\mathtt{wm}}$ (resp. $\to_{\mathtt{wgcv}}$; resp. $\to_{\mathtt{wm}}$).*

▶ **Theorem 7** (Confluence). *Reductions $\to_{\mathtt{w}}$ and $\to_{\mathtt{w}\neg\mathtt{gcv}}$ are confluent.*

Proof. By Newman lemma, local confluence (Lemma 5) and local termination (Prop. 4) imply that $\to_{\mathtt{wm}}$, $\to_{\mathtt{we}}$, and $\to_{\mathtt{wgcv}}$ are confluent separately. By a result of Hindley [36], strict strong local commutation (Lemma 6) and local termination imply that $\to_{\mathtt{wm}}$, $\to_{\mathtt{we}}$, and $\to_{\mathtt{wgcv}}$ are pairwise commuting. By Hindley-Rosen lemma, $\to_{\mathtt{w}} = \to_{\mathtt{wm}} \cup \to_{\mathtt{we}} \cup \to_{\mathtt{wgcv}}$ and $\to_{\mathtt{w}\neg\mathtt{gcv}} = \to_{\mathtt{wm}} \cup \to_{\mathtt{we}}$ are confluent. ◀

▶ Remark 8. The design choice that values are only abstraction is motivated by the commutation of $\to_{\mathtt{we}}$ over $\to_{\mathtt{wgcv}}$. Indeed, if one considers variables as values (thus allowing the erasure of variables by $\to_{\mathtt{gcv}}$), such a commutation fails, and confluence does not hold, as the following non-commuting (and non-confluent) span shows: $x[z\leftarrow ww]\ {}_{\mathtt{wgcv}}\!\leftarrow x[y\leftarrow z][z\leftarrow ww] \to_{\mathtt{e}_W} x[y\leftarrow ww][z\leftarrow ww]$.

Interestingly, something similar happens in CbNeed, where values are only abstractions too. Indeed, if one considers variables as values (thus allowing the duplication of variables in CbNeed), then one has the following non-closable critical pair for the CbNeed strategy, non-closable because y is not needed in $\mathtt{I}(\lambda z.wx)[x\leftarrow y][y\leftarrow \mathtt{I}]$ (the diagram closes in the CbNeed *calculus* but the CbNeed *strategy* is not confluent):

$$\begin{array}{ccc} x(\lambda z.wx)[x\leftarrow y][y\leftarrow \mathtt{I}] & \longrightarrow & x(\lambda z.wx)[x\leftarrow \mathtt{I}][y\leftarrow \mathtt{I}] \\ \downarrow & & \vdots \\ y(\lambda z.wx)[x\leftarrow y][y\leftarrow \mathtt{I}] & \dashrightarrow & \mathtt{I}(\lambda z.wx)[x\leftarrow \mathtt{I}][y\leftarrow \mathtt{I}] \end{array}$$

5 Silly Multi Types

In this section, we introduce multi types and the silly multi type system.

Inception. The design of the silly type system is specular to the one for CbNeed by Accattoli et al. [12]. The CbNeed one tweaks the CbV system in the literature (due to Ehrhard [32]) by changing its rules for applications and ESs, which are the rules using multi-sets, as to accommodate CbN erasures. The silly one given here tweaks the CbN system in the literature (due to de Carvalho [31]) by changing the same rules to accommodate CbV erasures. In both cases, the underlying system is responsible for the duplication behavior. The desired erasure behavior is then enforced by changing the rules using multi-sets.

The silly system is in Fig. 2. It is the variant of the system for CbN in [12] (itself a reformulation of [31] tuned for weak evaluation) obtained by adding "$\uplus$[norm]" in the right premise of both rules @ and ES. Such an extra typing for these rules captures the *silly extra copy* mentioned at the end of Sect. 2. More details are given at the end of this section.

$$\begin{array}{rrcl} \text{Linear types} & L, L' & ::= & \mathtt{norm} \mid M \to L \\ \text{Multi types} & M, N & ::= & [L_i]_{i\in I} \text{ where } I \text{ is a finite set} \\ \text{Generic types} & T, T' & ::= & L \mid M \end{array}$$

$$\frac{}{x:[L] \vdash^{(0,1)} x:L}\ \mathsf{ax} \qquad \frac{(\Gamma_i \vdash^{(m_i,e_i)} t:L_i)_{i\in I}}{\uplus_{i\in I}\Gamma_i \vdash^{(\sum_{i\in I} m_i, \sum_{i\in I} e_i)} t:[L_i]_{i\in I}}\ \mathsf{many}$$

$$\frac{}{\vdash^{(0,0)} \lambda x.t : \mathtt{norm}}\ \mathsf{ax}_\lambda \qquad \frac{\Gamma \vdash^{(m,e)} t : M \to L \qquad \Delta \vdash^{(m',e')} s : M \uplus [\mathtt{norm}]}{\Gamma \uplus \Delta \vdash^{(m+m'+1, e+e')} ts : L}\ @$$

$$\frac{\Gamma \vdash^{(m,e)} t:L}{\Gamma \setminus\!\setminus x \vdash^{(m,e)} \lambda x.t : \Gamma(x) \to L}\ \lambda \qquad \frac{\Gamma \vdash^{(m,e)} t:L \qquad \Delta \vdash^{(m',e')} s : \Gamma(x) \uplus [\mathtt{norm}]}{(\Gamma \setminus\!\setminus x) \uplus \Delta \vdash^{(m+m', e+e')} t[x \leftarrow s] : L}\ \mathsf{ES}$$

Figure 2 The silly multi type system.

CbS Types and Judgements. *Linear types* and *multi(-sets) types* are defined by mutual induction in Fig. 2. Note the linear constant $\mathtt{norm}$ used to type abstractions, which are normal terms. For conciseness, sometimes we shorten it to $\mathtt{n}$. We shall show that every normalizing term is typable with $\mathtt{norm}$, hence its name. The constant $\mathtt{norm}$ shall also play a role in our quantitative study in Sect. 9. The empty multi set $[\,]$ is also noted $\mathbf{0}$.

A multi type $[L_1, \dots, L_n]$ has to be intended as a conjunction $L_1 \wedge \cdots \wedge L_n$ of linear types $L_1, \dots, L_n$, for a commutative, associative, non-idempotent conjunction $\wedge$ (morally a tensor $\otimes$), of neutral element $\mathbf{0}$. The intuition is that a linear type corresponds to a single use of a term t, and that t is typed with a multiset M of n linear types if it is going to be used (at most) n times, that is, if t is part of a larger term s, then a copy of t shall end up in evaluation position during the evaluation of s.

Judgments have shape $\Gamma \vdash^{(m,e)} t : T$ where t is a term, m and e are two natural numbers, T is either a multi type or a linear type, and Γ is a *type context*, i.e., a total function from variables to multi types such that $\mathtt{dom}(\Gamma) := \{x \mid \Gamma(x) \neq \mathbf{0}\}$ is finite, usually written as $x_1 : M_1, \dots, x_n : M_n$ (with $n \in \mathbb{N}$) if $\mathtt{dom}(\Gamma) \subseteq \{x_1, \dots, x_n\}$ and $\Gamma(x_i) = M_i$ for $1 \leq i \leq n$.

The indices m and e shall be used for measuring the length of evaluation sequences via type derivations, namely to measure the number of $\to_{\mathtt{wm}}$ and $\to_{\mathtt{we}}$ steps. There is no index for $\to_{\mathtt{wgcv}}$ steps in order to stay close to the type systems in Accattoli et al. [12] for CbN/CbV/CbNeed, that do not have an index for GC either; the reason being that GC (by value) can be postponed. A quick look to the typing rules shows that m and e are not really needed, as m can be recovered as the number of $\mathtt{app}$ rules, and e as the number of $\mathtt{ax}$ rules. It is however handy to note them explicitly.

We write $\Gamma \vdash t : T$ when the information given by m and e is not relevant.

Typing Rules. The abstraction rule λ uses the notation $\Gamma \setminus\!\setminus x$ for the type context defined as Γ on every variable but possibly x, for which $(\Gamma \setminus\!\setminus x)(x) = \mathbf{0}$. It is a compact way to express the rule in both the cases $x \in \mathtt{dom}(\Gamma)$ and $x \notin \mathtt{dom}(\Gamma)$.

Rules @ and ES require the argument to be typed with $M \uplus [\mathtt{norm}]$, which is necessarily introduced by rule many, the hypotheses of which are a multi set of derivations, indexed by a possibly empty set I. When I is empty, the rule has one premises, the one for $\mathtt{norm}$.

Type Derivations. We write $\pi \triangleright \Gamma \vdash t : L$ if π is a (*type*) *derivation* (i.e. a tree constructed using the rules in Fig. 2) of final judgment $\Gamma \vdash t : L$.

The *size* $|\pi|$ of a derivation $\pi \triangleright \Gamma \vdash^{(m,e)} t : A$ is the number of non-many rules, which is always greater or equal to the sum of the indices, that is, $|\pi| \geq m + e$.

Further Technicalities about Types. The type context Γ is *empty* if $\mathtt{dom}(\Gamma) = \emptyset$, and we write $\vdash t : L$ when Γ is empty. *Multi-set sum* $\uplus$ is extended to type contexts point-wise, i.e. $(\Gamma \uplus \Delta)(x) := \Gamma(x) \uplus \Delta(x)$ for each variable x. This notion is extended to a finite family of type contexts as expected, in particular $\uplus_{i\in J}\Gamma_i$ is the empty context when $J = \emptyset$. Given two type contexts Γ and Δ such that $\mathtt{dom}(\Gamma) \cap \mathtt{dom}(\Delta) = \emptyset$, the type context Γ, Δ is defined by $(\Gamma, \Delta)(x) := \Gamma(x)$ if $x \in \mathtt{dom}(\Gamma)$, $(\Gamma, \Delta)(x) := \Delta(x)$ if $x \in \mathtt{dom}(\Delta)$, and $(\Gamma, \Delta)(x) := \mathbf{0}$ otherwise. Note that $\Gamma, x : \mathbf{0} = \Gamma$, where we implicitly assume $x \notin \mathtt{dom}(\Gamma)$.

Relevance. Note that no weakening is allowed in axioms. An easy induction then shows:

▶ **Lemma 9** (Type contexts and variable occurrences). *Let* $\pi \triangleright \Gamma \vdash^{(m,e)} t : L$ *be a derivation. Then* $\mathtt{shfv}(t) \subseteq \mathtt{dom}(\Gamma) \subseteq \mathtt{fv}(t)$.

Lemma 9 implies that derivations of closed terms have empty type context. Note that free variables of t might not be in $\mathtt{dom}(\Gamma)$, if they only occurr in abstractions typed with ax_λ.

Typing the Silly Extra Copy. The empty multi type $\mathbf{0}$ is the type for variables that do not occur or whose occurrences are unreachable by weak evaluation. A typical example is $\lambda x.y$, that can be typed only with arrow types of the form $\mathbf{0} \to L$ (plus of course with $\mathtt{norm}$), because of Lemma 9. Note that in the silly system every term – even diverging ones – can be typed with $\mathbf{0}$ by rule many (taking 0 premises). In CbN, an argument for $\lambda x.y$ would only need to be typed with $\mathbf{0}$, as typability with $\mathbf{0}$ means that the term shall be erased. In the silly system, instead, the application rule @ requires an argument of $\lambda x.y$ to additionally be typed with $[\mathtt{norm}]$, because of the "$\uplus[\mathtt{norm}]$" requirement for arguments (and ESs), forcing the argument to be $\to_{\mathtt{w}}$ normalizing and capturing silly erasures.

Now, note that the modification $\uplus[\mathtt{norm}]$ at work in the silly system concerns *all* arguments and ESs, not only those associated to $\mathbf{0}$. This is what correspond at the type level to the reduction of the *silly extra copy* of *every* argument/ES (out of abstractions).

Relationship with the Literature. Our system is essentially the one used by Kesner and Ventura to measure CbN strong normalization in the LSC with respect to strong evaluation, i.e. possibly under abstraction [42]. There are two differences. Firstly, they tweak the @ and ES CbN rules with $\uplus[L]$ rather than with $\uplus[\mathtt{norm}]$, that is, they allow an arbitrary linear type for the additional copy to be evaluated. Secondly, they do not have rule ax_λ, because their evaluation is strong. Thus, $\mathtt{I}[x \leftarrow \lambda y.\Omega]$ is not typable in [42], while here it is.

The CbV system by Manzonetto et al. [46] is similar to ours in that it tweaks the CbN system and requires multi-sets to be non-empty.

6 The Weak Calculus, Types, and Strong Normalization

Here, we show that silly multi types characterize strong normalization in the weak SSC. The proof technique is standard: we prove correctness – i.e. t typable implies $t \in \mathrm{SN}_{\mathtt{w}}$ – via subject reduction, and completeness – i.e. $t \in \mathrm{SN}_{\mathtt{w}}$ implies t typable – via subject expansion and typability of normal forms. Note that SN in our weak case is simpler than SN in strong calculi, since here erasing steps cannot erase divergence. Actually, they cannot erase *any* step, as guaranteed by strict commutation in Sect. 4. At the end of the section, we shall indeed obtain *uniform normalization*, i.e. that weak and strong normalization for $\to_{\mathtt{w}}$ coincide.

Correctness. As it is standard, subject reduction for $\to_{\mathtt{we}}$ is based on a linear substitution lemma, in the technical report [14].

▶ **Proposition 10** (Quantitative subject reduction for Weak SSC). *Let $\pi \triangleright \Gamma \vdash^{(m,e)} t : L$ be a derivation.*

1. Multiplicative*: if $t \to_{\mathtt{wm}} s$ then $m \geq 1$ and there exists $\rho \triangleright \Gamma \vdash^{(m',e)} s : L$ with $m > m'$.*
2. Exponential*: if $t \to_{\mathtt{we}} s$ then $e \geq 1$ and there exists $\rho \triangleright \Gamma \vdash^{(m,e')} s : L$ with $e > e'$.*
3. GC by value*: if $t \to_{\mathtt{wgcv}} s$ then there exists $\rho \triangleright \Gamma \vdash^{(m,e)} s : L$ with $|\pi| > |\rho|$.*

Note that $\to_{\mathtt{wgcv}}$ steps do not change the m and e indices. This is a consequence of rules @ and ES having been modified for CbS with $\uplus[\mathtt{norm}]$ (and not with $\uplus[L]$ as in [42]).

▶ **Theorem 11** (Weak SSC correctness). *Let t be a term. If $\pi \triangleright \Gamma \vdash^{(m,e)} t : L$ then $t \in SN_{\mathtt{w}}$. Moreover, if $d : t \to_{\mathtt{w}}^{*} n$ is a normalizing sequence then $|d|_{\mathtt{wm}} \leq m$ and $|d|_{\mathtt{we}} \leq e$.*

Proof. By lexicographic induction on $(m + e, |\pi|)$ and case analysis on whether t reduces or not. If t is $\to_{\mathtt{w}}$-normal then the statement trivially holds. If t is not $\to_{\mathtt{w}}$-normal we show that all its reducts are SN for $\to_{\mathtt{w}}$, that is, t is SN. If $t \to_{\mathtt{wm}} u$ then by quantitative subject reduction (Prop. 28) there is a derivation $\rho \triangleright \Gamma \vdash^{(m',e)} u : L$ with $m' < m$. By *i.h.*, u is SN. If $t \to_{\mathtt{we}} u$ or $t \to_{\mathtt{wgcv}} u$ we reason similarly, looking at the e index for $\to_{\mathtt{we}}$ and to the size $|\rho|$ of the derivation for $\to_{\mathtt{wgcv}}$. The *moreover* part follows from the fact that $\to_{\mathtt{wm}}$ (resp. $\to_{\mathtt{we}}$) steps strictly decrease m (resp. e). ◀

Completeness. For completeness, we first need typability of weak normal forms. The two points of the next proposition are proved by mutual induction. The second point is stronger, as it has a universal quantification about linear types, crucial for the induction to go through.

▶ **Proposition 12** (Weak normal forms are typable).

1. *Let a be a weak answer. Then there exists $\pi \triangleright \Gamma \vdash^{(0,0)} a : \mathtt{norm}$ with $\mathtt{dom}(\Gamma) = \mathtt{shfv}(a)$.*
2. *Let i be an inert term. Then for any linear type L there exist a type context Γ such that $\mathtt{dom}(\Gamma) = \mathtt{shfv}(i)$ and a derivation $\pi \triangleright \Gamma \vdash^{(0,0)} i : L$.*

The rest of the proof of completeness is dual to the one for correctness, with an anti-substitution lemma (in the technical report [14]) needed for subject expansion for $\to_{\mathtt{we}}$. We omit the indices because for completeness they are irrelevant.

▶ **Proposition 13** (Subject expansion for Weak SSC). *Let $\pi \triangleright \Gamma \vdash s : L$ be a derivation. If $t \to_{\mathtt{w}} s$ then there exists a derivation $\rho \triangleright \Gamma \vdash t : L$.*

▶ **Theorem 14** (Weak SSC completeness). *Let t be a term. If $t \to_{\mathtt{w}}^{*} n$ and n is a weak normal form then there exists $\pi \triangleright \Gamma \vdash t : \mathtt{norm}$.*

Proof. By induction on $k = |d|$. If $k = 0$: then $t = n$ and is typable with $\mathtt{norm}$ by Prop. 12. If $k > 0$ then $t \to_{\mathtt{w}} u \to_{\mathtt{w}}^{k-1} n$ for some u and by *i.h.* there is a derivation $\pi' \triangleright \Gamma \vdash u : L$. By subject expansion (Prop. 13), $\pi \triangleright \Gamma \vdash t : L$. ◀

▶ **Corollary 15** (Weak SSC Uniform normalization). *t is weakly $\to_{\mathtt{w}}$-normalizing if and only if t is strongly $\to_{\mathtt{w}}$-normalizing.*

Proof. The non-obvious direction is $\Rightarrow$. By completeness (Th. 14), $t \in \mathrm{WN}_{\mathtt{w}}$ implies typability, which in turn, by correctness (Th. 11), implies $t \in \mathrm{SN}_{\mathtt{w}}$. ◀

Terms	t, s, u	$::=$	$x \mid \lambda x.t \mid ts$	Rewriting rule	
Values	v, v'	$::=$	$\lambda x.t$	$(\lambda x.t)v$	$\mapsto_{\beta_v} \quad t\{x \leftarrow v\}$
CbV contexts	V	$::=$	$\langle\cdot\rangle \mid tV \mid Vt$	$\rightarrow_{\beta_v}$	$:= \quad V\langle\mapsto_{\beta_v}\rangle$

Figure 3 The call-by-value λ-calculus.

7 Call-by-Value and Operational Equivalence

Here, we show that the silly multi types characterize CbV termination as well, and infer the operational equivalence of the SSC and CbV.

Closed CbV without ESs. We define the CbV λ-calculus in Fig. 3, mostly following the presentation of Dal Lago and Martini [29], for which the β_v-rule is non-deterministic but diamond (thus trivially uniformly normalizing, see the rewriting preliminaries). The only change with respect to [29] is that here values are only abstractions, for uniformity with the Weak SSC. We shall consider the weak evaluation of closed terms only, for which there is no difference whether variables are values or not, since free variables cannot be arguments (out of abstractions) anyway. Closed normal forms are exactly the abstractions.

We discuss only the closed case for CbV because the silly type system is *not* correct for CbV with open terms. This point is properly explained after the operational equivalence theorem. It is also the reason why we do not present CbV via the LSC, as also explained after the theorem. The problem with open terms does not hinder the operational equivalence of the SSC and CbV, because contextual equivalence is based on closed terms only.

Judgements for CbV. In this section, the index e of the silly type system does not play any role, because $\rightarrow_{\beta_v}$ steps are bound only by the m index. Therefore, we omit e and write $\pi \triangleright \Gamma \vdash^m t : L$ instead of $\pi \triangleright \Gamma \vdash^{(m,e)} t : L$.

Correctness. The proof technique is the standard one. As usual, subject reduction is proved via a substitution lemma specified in the technical report [14]. The index m is used as decreasing measure to prove correctness.

▶ **Proposition 16** (Quantitative subject reduction for Closed CbV). *Let t be a closed term. If $\pi \triangleright \vdash^m t : L$ and $t \rightarrow_{\beta_v} s$ then $m \geq 1$ and there exists $\rho \triangleright \vdash^{m'} s : L$ with $m > m'$.*

▶ **Theorem 17** (Closed CbV correctness). *Let t be a closed term. If $\pi \triangleright \vdash^m t : L$ then there are an abstraction v and a reduction sequence $d\colon t \rightarrow^*_{\beta_v} v$ with $|d| \leq m$.*

Completeness. Completeness is also proved in a standard way, omitting the index m because it is irrelevant.

▶ **Proposition 18** (Subject expansion for Closed CbV). *If $\pi \triangleright \vdash s : L$ and $t \rightarrow_{\beta_v} s$ then and there exists a typing ρ such that $\rho \triangleright \vdash t : L$.*

▶ **Theorem 19** (Closed CbV completeness). *Let t be a closed λ-term. If there exists a value v and a reduction sequence $d\colon t \rightarrow^*_{\beta_v} v$ then $\pi \triangleright \vdash t : \mathtt{norm}$.*

Operational Equivalence. We define abstractly contextual equivalence for a language of terms and arbitrary contexts which are terms with an additional hole construct.

▶ **Definition 20** (Contextual Equivalence). *Given a rewriting relation $\to$, we define the associated* contextual equivalence $\simeq_{\mathtt{c}}$ *as follows: $t \simeq_{\mathtt{c}} t'$ if, for all contexts C such that $C\langle t\rangle$ and $C\langle t'\rangle$ are closed terms, $C\langle t\rangle$ is weakly $\to$-normalizing iff $C\langle t'\rangle$ is weakly $\to$-normalizing.*

Let $\simeq_C^{\mathtt{silly}}$ and $\simeq_C^{\mathtt{value}}$ be the contextual equivalences for $\to_{\mathtt{w}}$ and $\to_{\beta_v}$. The next section shall show that $\simeq_C^{\mathtt{silly}}$ can equivalently be defined using the CbS strategy.

▶ **Theorem 21** (Operational equivalence of CbS and CbV). *On λ-terms, $t \simeq_C^{\mathtt{silly}} s$ iff $t \simeq_C^{\mathtt{value}} s$.*

Proof. On closed λ-terms, both $\to_{\mathtt{w}}$-termination and $\to_{\beta_v}$-termination are equivalent to typability in the silly system (Theorems 11 and 14 for $\to_{\mathtt{y}}$ and Theorems 17 and 19 for $\to_{\beta_v}$). Thus the contextual equivalences coincide. ◀

Call-by-Silly Helps to Prove Contextual Equivalence. As the last theorem says, contextual equivalences induced by CbV and CbS coincide. Even though they equate the same terms, CbS reduction sometimes provides a way to prove CbV contextual equivalence in cases where CbV does not. Consider the following four different terms, where i could be any normal form that is not of the shape $S\langle v\rangle$, for example $i = y\mathtt{I}$:

$$(\lambda x.xx)\, i \qquad (\lambda x.xi)\, i \qquad (\lambda x.ii)\, i \qquad ii$$

These four terms can intuitively be seen as CbV contextually equivalent, as we now outline. When one of these terms is plugged in a closing context C, reduction shall provide substitutions on i making it either converge to a value v or diverge. if it diverges, so will the four terms. If it converges to v, then all four terms will reduce to vv. This reasoning however cannot easily be made formal.

The easiest way to prove that two terms are contextually equivalent for a reduction $\to_r$ is to prove that they are related by $=_r$, the smallest equivalence relation including $\to_r$. We shall now see how the silly calculus helps in equating more terms (than CbV) with its reduction.

The four terms above are not $=_{\beta_v}$-related. First, note that the four terms are all $\to_{\beta_v}$-normal (assuming that i is $\to_{\beta_v}$-normal). If they were $=_{\beta_v}$-related, then by the Church-Rosser property they should have a common reduct. As the four terms are syntactically different normal forms, they cannot be equated by $=_{\beta_v}$.

The first three terms are $=_{\mathtt{w}}$-related. The first three terms rewrite in the SSC to $ii[x{\leftarrow}i]$, which is why $(\lambda x.xx)\, i =_{\mathtt{w}} (\lambda x.xi)\, i =_{\mathtt{w}} (\lambda x.ii)\, i =_{\mathtt{w}} ii[x{\leftarrow}i]$.

Unfortunately, the fourth term ii is a $\to_{\mathtt{w}}$-normal form and hence $ii \neq_{\mathtt{w}} ii[x{\leftarrow}i]$. Thus, not all CbV contextually equivalent terms are equated by the silly calculus, not even when the difference amounts to the duplication of a non-value term. The issue has to do with the *silly extra copy* that cannot be erased easily. We believe that a small-step silly calculus could help equating more terms, but we have not managed to work out multi types for such a calculus.

We refer the reader to the technical report [14] for the proof that $=_{\mathtt{w}}$-related terms are contextually equivalent.

The Issue with Open CbV and the Silly Type System. There is an issue if one considers the silly type system relatively to CbV with open terms, namely subject reduction breaks. This point is delicate. In fact, there are no issues if one considers only Plotkin's β_v rule, except that Plotkin's rule is *not* an adequate operational semantics for CbV with open terms, as it is well-known and discussed at length by Accattoli and Guerrieri [10, 11]. Adequate

operational semantics for Open CbV do extend Plotkin's. One such semantics is Carraro and Guerrieri's *shuffling calculus* [27], that extends β_v by adding some σ-rules. We briefly discuss it here; the last paragraph of this section shall explain because we prefer it to the CbV LSC for the explanation of the issue with open terms.

It turns out that one of the σ rules breaks subject reduction for the silly type system (while there are no problems if one considers instead Ehrhard's CbV multi types [32]), as we now show. Rule $\mapsto_{\sigma_3}$ is defined as follows:

$$z((\lambda x.y)s) \quad \mapsto_{\sigma_3} \quad (\lambda x.zy)s$$

For $n \geq 1$, we have the following derivation for the source term $z((\lambda x.y)s)$ in the silly type system:

$$\dfrac{\pi_z \triangleright \dots \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{y:[A]\vdash y:A}\,\mathsf{ax}}{y:[A]\vdash \lambda x.y:\mathbf{0}\to A}\,\lambda \qquad \dfrac{\pi_s \triangleright \Gamma \vdash s:\mathtt{norm}}{\Gamma \vdash s:[\mathtt{norm}]}\,\mathsf{many}}{(y:[A],\Gamma\vdash(\lambda x.y)s:A)_{i=1,\dots,n}}\,@ \qquad \dfrac{\dfrac{\dfrac{}{y:[\mathtt{norm}]\vdash y:\mathtt{norm}}\,\mathsf{ax}}{y:[\mathtt{norm}]\vdash \lambda x.y:\mathbf{0}\to\mathtt{norm}}\,\lambda \qquad \dfrac{\pi_s \triangleright \Gamma \vdash s:\mathtt{norm}}{\Gamma \vdash s:[\mathtt{norm}]}\,\mathsf{many}}{y:[\mathtt{norm}],\Gamma\vdash(\lambda x.y)s:\mathtt{norm}}\,@}{y:[A^n,\mathtt{norm}],\Gamma^{n+1}\vdash(\lambda x.y)s:[A^n,\mathtt{norm}]}\,\mathsf{many}}{z:[[A^n,\mathtt{norm}]\to B],y:[A^n,\mathtt{norm}],\Gamma^{n+1}\vdash z((\lambda x.y)s):B}\,@$$

where $\pi_z \triangleright \dots$ stands for:

$$\dfrac{}{\pi_z \triangleright z:[[A^n]\to B]\vdash z:[A^n]\to B}\,\mathsf{ax}$$

The target term $(\lambda x.zy)s$ of rule $\mapsto_{\sigma_3}$, instead, can only be typed as follows, the key point being that Γ^{n+1} is replaced by Γ:

$$\dfrac{\dfrac{\dfrac{}{z:[[A^n]\to B]\vdash z:[A^n]\to B}\,\mathsf{ax} \qquad \dfrac{\dfrac{}{(y:[A]\vdash y:A)_{i=1,\dots,n}}\,\mathsf{ax} \qquad \dfrac{}{y:[\mathtt{norm}]\vdash y:\mathtt{norm}}\,\mathsf{ax}}{y:[A^n,\mathtt{norm}]\vdash y:[A^n,\mathtt{norm}]}\,\mathsf{many}}{z:[[A^n]\to B],y:[A^n,\mathtt{norm}]\vdash zy:B}\,@}{z:[[A^n]\to B],y:[A^n,\mathtt{norm}]\vdash \lambda x.zy:\mathbf{0}\to B}\,\lambda \qquad \dfrac{\pi_s \triangleright \Gamma\vdash s:\mathtt{norm}}{\Gamma\vdash s:[\mathtt{norm}]}\,\mathsf{many}}{z:[[A^n]\to B],y:[A^n,\mathtt{norm}],\Gamma\vdash(\lambda x.zy)s:B}\,@$$

This counter-example adapt the counter-example given by Delia Kesner to subject reduction for the multi type system by Manzonetto et al. [46, 37], as reported in the long version on Arxiv of [11], which appeared after the publication of [46, 37], where there is no mention of this issue. The work in the present paper can be actually seen as a clarification of the failure of subject reduction for the system in [46, 37]. Essentially, the system in [46, 37] is a system for CbS, not for CbV, but is therein used to study CbV strong evaluation with possibly open terms, unaware that the system models a different evaluation mechanism.

We conjecture that the silly type system is adequate for Open CbV (that is, a term is silly typable if and only if it is CbV terminating) even if it is not invariant for Open CbV (that is, subject reduction does not hold).

Naturality of the Issue. A first reaction to the shown issue is to suspect that something is wrong or ad-hoc in our approach, especially given that the operational equivalence of CbN and CbNeed does not suffer of this issue, that is, the CbN multi type system is invariant for CbNeed evaluation of open terms. At high-level, however, the issue is natural and to be expected, as we now explain. The two systems of each pair CbN/CbNeed and CbS/CbV have different duplicating policies and the same erasing policy. The pair CbN/CbNeed has no restrictions on erasure, thus open normal forms have no garbage. Therefore, the different ways in which they duplicate garbage are not observable. For the pair CbS/CbV, instead, erasure is restricted to *values*, with the consequence that garbage has to be evaluated before possibly being erased, and that with open terms some garbage might never be erased. Thus, the different duplicating policies of CbS/CbV leave different amounts of non-erasable garbage in open normal forms, which is observable and changes the denotational semantics.

The Open CbV LSC. Another adequate formalism for Open CbV is the VSC, or its micro-step variant, the CbV LSC. We now discuss the CbV LSC, but everything we say applies also to the VSC.

The difference between the shuffling calculus and the CbV LSC is that the latter uses ESs and modifies the rewriting rules at a distance. In particular, duplication is done by the following exponential rule:

$$W\langle x\rangle[x{\leftarrow}S\langle v\rangle] \quad \to_{\mathtt{ve}} \quad S\langle W\langle v\rangle[x{\leftarrow}v]\rangle$$

Subject reduction breaks also for the CbV LSC, as we can show by adapting Kesner's counter-example. Consider the step $(zy')[y'{\leftarrow}\mathtt{I}[y{\leftarrow}s]] \to_{\mathtt{ve}} (z\mathtt{I})[y'{\leftarrow}\mathtt{I}][y{\leftarrow}s]$:

$$\dfrac{\pi_1 \rhd z : [[A^n]{\to}B],\, y' : [A^n] \vdash zy' : B \qquad \dfrac{\dfrac{\dfrac{\pi_s \rhd \Gamma \vdash s : \mathtt{norm}}{\Gamma \vdash s : [\mathtt{norm}]}\,\mathsf{many} \qquad \pi_{\mathtt{I}} \vdash \mathtt{I} : A}{(\Gamma \vdash \mathtt{I}[y{\leftarrow}s] : A)_{i=1,\dots,n}}\,\mathsf{ES} \qquad \dfrac{\dfrac{\pi_s \rhd \Gamma \vdash s : \mathtt{norm}}{\Gamma \vdash s : [\mathtt{norm}]}\,\mathsf{many} \qquad \dfrac{}{\vdash \mathtt{I} : \mathtt{norm}}\,\mathsf{ax}_\lambda}{\Gamma \vdash \mathtt{I}[y{\leftarrow}s] : \mathtt{norm}}\,\mathsf{ES}}{\Gamma^{n+1} \vdash \mathtt{I}[y{\leftarrow}s] : [A^n, \mathtt{norm}]}\,\mathsf{many}}{z : [[A^n]{\to}B],\, \Gamma^{n+1} \vdash (zy')[y'{\leftarrow}\mathtt{I}[y{\leftarrow}s]] : B}\,\mathsf{ES}$$

As for $\mapsto_{\sigma_3}$, the key point is that that Γ^{n+1} gets replaced by Γ in the typing of the reduct:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{z : [[A^n]{\to}B] \vdash z : [A^n]{\to}B}\,\mathsf{ax} \qquad \dfrac{(\pi_{\mathtt{I}} \vdash \mathtt{I} : A)_{i=1,\dots,n} \qquad \dfrac{}{\vdash \mathtt{I} : \mathtt{norm}}\,\mathsf{ax}_\lambda}{\vdash \mathtt{I} : [A^n, \mathtt{norm}]}\,\mathsf{many}}{z : [[A^n]{\to}B] \vdash z\mathtt{I} : B}\,\mathsf{ES} \qquad \dfrac{\dfrac{}{\vdash \mathtt{I} : \mathtt{norm}}\,\mathsf{ax}_\lambda}{\vdash \mathtt{I} : [\mathtt{norm}]}\,\mathsf{many}}{z : [[A^n]{\to}B] \vdash (z\mathtt{I})[y'{\leftarrow}\mathtt{I}] : B}\,\mathsf{ES} \qquad \dfrac{\pi_s \rhd \Gamma \vdash s : \mathtt{norm}}{\Gamma \vdash s : [\mathtt{norm}]}\,\mathsf{many}}{z : [[A^n]{\to}B],\, \Gamma \vdash (z\mathtt{I})[y'{\leftarrow}\mathtt{I}][y{\leftarrow}s] : B}\,\mathsf{ES}$$

What breaks it is the use of the substitution context S in $\to_{\mathtt{ve}}$, which can be seen as the analogous of rule $\mapsto_{\sigma_3}$ of the shuffling calculus. The difference is that while $\mapsto_{\sigma_3}$ is needed only for open terms in the shuffling calculus, rule $\to_{\mathtt{ve}}$ is used also for the evaluation of closed terms in the CbV LSC. This is why we preferred to avoid using the CbV LSC to study Closed CbV. In fact, one can prove that in the closed case the modification of rule $\to_{\mathtt{ve}}$ without S is enough to reach normal forms. But this fact needs a technical study and a theorem, which we preferred to avoid.

8 The Call-by-Silly Strategy

In this section, we define the CbS evaluation strategy as a sort of extension of the CbN one. We then start by recalling the CbN erasing rule and the CbN strategy.

CbN Erasure. The *garbage collection (GC) rule* $\mapsto_{\mathtt{gc}}$ in Fig. 4 eliminates the ES $t[x{\leftarrow}s]$ when the bound variable x does not occur in t. Rule $\mapsto_{\mathtt{gc}}$ is the CbN form of erasure and it is not part of the rules of the SSC; it is given here only to be able to define the CbN strategy. Example: $(\lambda z.yy)[x{\leftarrow}\mathtt{I}[w{\leftarrow}t]] \mapsto_{\mathtt{gc}} \lambda z.yy$.

Call-by-Name. In Fig. 4, we define the CbN strategy $\to_{\mathtt{n}}$ of the weak LSC, first appeared in Accattoli et al. [5]. The CbN strategy uses the CbN GC rule $\mapsto_{\mathtt{gc}}$. Note that CbN evaluation contexts N never enter into arguments or ESs. The CbN strategy is almost deterministic: rules $\to_{\mathtt{nm}}$ and $\to_{\mathtt{ne}}$ are deterministic and its erasing rule $\to_{\mathtt{ngc}}$ is non-deterministic but diamond; for instance $\mathtt{I}[z{\leftarrow}\mathtt{II}]_{\mathtt{ngc}}{\leftarrow}\mathtt{I}[y{\leftarrow}\mathtt{I}][z{\leftarrow}\mathtt{II}] \to_{\mathtt{ngc}} \mathtt{I}[y{\leftarrow}\mathtt{I}]$, and then $\mathtt{I}[z{\leftarrow}\mathtt{II}] \to_{\mathtt{ngc}} \mathtt{I}_{\mathtt{ngc}}{\leftarrow}\mathtt{I}[y{\leftarrow}\mathtt{I}]$. Moreover, $\to_{\mathtt{gc}}$ is postponable, i.e. if $t \to_{\mathtt{n}}^* s$ then $t \to_{\mathtt{n}\neg\mathtt{gc}}^* \to_{\mathtt{gc}}^* s$ where $\to_{\mathtt{n}\neg\mathtt{gc}} := \to_{\mathtt{nm}} \cup \to_{\mathtt{ne}}$, which is why it is often omitted from the micro-step presentation of CbN.

CALL-BY-NAME STRATEGY $\to_{\mathtt{n}}$

NAME CTXS $N, N' ::= \langle\cdot\rangle \mid Nt \mid N[x{\leftarrow}t]$	$\to_{\mathtt{nm}} := N\langle \mapsto_{\mathtt{m}} \rangle$
ROOT GC $t[x{\leftarrow}s] \mapsto_{\mathtt{gc}} t \quad \text{if } x \notin \mathtt{fv}(t)$	$\to_{\mathtt{ne}} := N\langle \mapsto_{\mathtt{e}_N} \rangle$
	$\to_{\mathtt{ngc}} := N\langle \mapsto_{\mathtt{gc}} \rangle$
	$\to_{\mathtt{n}} := \to_{\mathtt{nm}} \cup \to_{\mathtt{ne}} \cup \to_{\mathtt{ngcv}}$

CALL-BY-SILLY STRATEGY $\to_{\mathtt{y}}$

ANSWERS $a, a' ::= v \mid a[x{\leftarrow}a']$	$\to_{\mathtt{ym}} := Y\langle \mapsto_{\mathtt{m}} \rangle$
AUX. CTXS $A, A' ::= \langle\cdot\rangle \mid a[x{\leftarrow}A] \mid A[x{\leftarrow}t]$	$\to_{\mathtt{ygcv}} := Y\langle \mapsto_{\mathtt{gcv}} \rangle$
SILLY CTXS $Y, Y' ::= A\langle N \rangle$	$\to_{\mathtt{ye_{AY}}} := A\langle \mapsto_{\mathtt{e}_Y} \rangle$
	$\to_{\mathtt{ye_{YN}}} := Y\langle \mapsto_{\mathtt{e}_N} \rangle$
	$\to_{\mathtt{y}} := \to_{\mathtt{ym}} \cup \to_{\mathtt{ye_{AY}}} \cup \to_{\mathtt{ye_{YN}}} \cup \to_{\mathtt{ygcv}}$

Figure 4 The call-by-name and call-by-silly strategies.

The Call-by-Silly Strategy. The CbS strategy $\to_{\mathtt{y}}$ is defined in Fig. 4 via silly evaluation contexts Y – explained next – (we use Y for *sill*Y because S is already used for substitution contexts) and the CbV erasing rule $\mapsto_{\mathtt{gcv}}$ of the SSC. Letting GC aside, CbS is the extension of CbN that, once (non-erasing) CbN can no longer reduce, starts evaluating the ESs (out of abstractions) left hanging by the CbN strategy. Such an extension is specified via the *auxiliary contexts* A, whose key production is $t[x{\leftarrow}A]$ for evaluation contexts where t is a CbS normal form: evaluation enters an ESs only when it is sure that it is no longer needed. There are, however, a few subtleties.

Firstly, *no longer needed* does not necessarily mean *without free occurrences*, as occurrences might be blocked by abstractions, as for instance in $(\lambda y.x)[x{\leftarrow}s]$ where x occurs but only under abstraction, and the CbS has to evaluate inside s. Therefore, *no longer needed* rather means *without shallow free occurrences* (Def. 1, page 7).

Secondly, in order to be sure that no occurrences of x shall become shallow because of other steps, we shall ask that in $t[x{\leftarrow}Y]$ the sub-term t is a normal form, otherwise in a case such as $((\lambda y.x)\mathtt{I})[x{\leftarrow}s]$ the reduction of the multiplicative step to $x[y{\leftarrow}\mathtt{I}][x{\leftarrow}s]$ turns a blocked occurrence of x into a shallow occurrence. Because of rules at a distance, the term $(\lambda y.x)[x{\leftarrow}s]\mathtt{I}$ suffers of the same problem but the argument comes *on the right* of the ES. For this reason, we forbid the evaluation context to be applied once it enters an ES, that is, silly contexts Y are defined as $A\langle N\rangle$, that is, rigidly separating the construction that applies (at work in N) from the one that enters ES (at work in A).

Thirdly, such a rigid separation in the definition of silly contexts forces us to have *two* exponential rules. Exponential rules use contexts twice in their definition: to select the occurrence to replace in the root rule *and* to extend the applicability of the root rule. The subtlety is that if both these contexts are silly, then one can nest an A context (selecting an occurrence) under a N context (extending the rule), which, as explained, has to be avoided. Therefore there are two silly exponential rules $\to_{\mathtt{ye_{AY}}}$ and $\to_{\mathtt{ye_{YN}}}$ carefully designed as to avoid the dangerous combination (that would be given by $N\langle \mapsto_{\mathtt{e}_A} \rangle$).

Fourthly, in order to mimic the property of CbN that GC is postponable, we shall ask that in $t[x{\leftarrow}Y]$ the sub-term t is a normal form only for the *non-erasing* CbS, otherwise in a case such as $x[y{\leftarrow}\mathtt{I}][x{\leftarrow}s]$ the CbS strategy would be forced to erase $[y{\leftarrow}\mathtt{I}]$ before evaluating s. Normal forms for non-erasing CbS are characterized below (Prop. 23) exactly as the answers defined in Fig. 4. We shall also prove that $\to_{\mathtt{ygcv}}$ steps are postponable (Prop. 3).

The CbS strategy is supposed to be applied to closed terms only, while there is no closure hypothesis on the weak calculus.

▶ **Example 22.** A good example to observe the differences between CbN and CbS is given by the term $t := (\lambda y.\lambda x.x(\lambda w.x))\Omega(\mathtt{II})$ where $\mathtt{I} := \lambda z.z$, $\Omega := \delta\delta$, and $\delta := \lambda z.zz$. In CbN, it evaluates with 4 multiplicative steps and 3 exponential steps, as follows:

$$\begin{array}{lll} t & \to_{\mathtt{nm}} (\lambda x.x(\lambda w.x))[y{\leftarrow}\Omega](\mathtt{II}) & \to_{\mathtt{nm}} (x(\lambda w.x))[x{\leftarrow}\mathtt{II}][y{\leftarrow}\Omega] \\ & \to_{\mathtt{ne}} ((\mathtt{II})(\lambda w.x))[x{\leftarrow}\mathtt{II}][y{\leftarrow}\Omega] & \to_{\mathtt{nm}} (z[z{\leftarrow}\mathtt{I}](\lambda w.x))[x{\leftarrow}\mathtt{II}][y{\leftarrow}\Omega] \\ & \to_{\mathtt{ne}} (\mathtt{I}[z{\leftarrow}\mathtt{I}](\lambda w.x))[x{\leftarrow}\mathtt{II}][y{\leftarrow}\Omega] & \to_{\mathtt{nm}} z'[z'{\leftarrow}\lambda w.x][z{\leftarrow}\mathtt{I}][x{\leftarrow}\mathtt{II}][y{\leftarrow}\Omega] \\ & \to_{\mathtt{ne}} (\lambda w.x)[z'{\leftarrow}\lambda w.x][z{\leftarrow}\mathtt{I}][x{\leftarrow}\mathtt{II}][y{\leftarrow}\Omega] & =: s \end{array}$$

The obtained term s is normal for the multiplicative and exponential rules. Since in CbN one can erase every term, s then reduces (in CbN) as follows:

$$s \quad \to_{\mathtt{ngc}} \quad (\lambda w.x)[z{\leftarrow}\mathtt{I}][x{\leftarrow}\mathtt{II}][y{\leftarrow}\Omega] \quad \to_{\mathtt{ngc}} \quad (\lambda w.x)[x{\leftarrow}\mathtt{II}][y{\leftarrow}\Omega] \quad \to_{\mathtt{ngc}} \quad (\lambda w.x)[x{\leftarrow}\mathtt{II}]$$

The obtained term is now normal for the CbN strategy, since evaluation does not enter into ESs. Note that t diverges for CbV evaluation (defined in Sect. 7). In CbS, the first part of the evaluation of t, up to s, is the same as for CbN (where $\to_{\mathtt{ne}}$ steps become $\to_{\mathtt{ye_{YN}}}$ steps). Then, s evaluates differently than in CbN, diverging:

$$\begin{array}{llll} s & \to_{\mathtt{ygcv}} (\lambda w.x)[z{\leftarrow}\mathtt{I}][x{\leftarrow}\mathtt{II}][y{\leftarrow}\Omega] & \to_{\mathtt{ygcv}} (\lambda w.x)[x{\leftarrow}\mathtt{II}][y{\leftarrow}\Omega] & \\ & \to_{\mathtt{ym}} (\lambda w.x)[x{\leftarrow}z[z{\leftarrow}\mathtt{I}]][y{\leftarrow}\Omega] & \to_{\mathtt{ye_{AY}}} (\lambda w.x)[x{\leftarrow}\mathtt{I}[z{\leftarrow}\mathtt{I}]][y{\leftarrow}\Omega] & \\ & \to_{\mathtt{ygcv}} (\lambda w.x)[x{\leftarrow}\mathtt{I}][y{\leftarrow}\Omega] & \to_{\mathtt{ym}} (\lambda w.x)[x{\leftarrow}\mathtt{I}][y{\leftarrow}zz[z{\leftarrow}\delta]] & \ldots \end{array}$$

Rewriting Properties of CbS. We start by characterizing normal form for both the CbS strategy and its non-erasing variant $\to_{\mathtt{y}\neg\mathtt{gcv}} := \to_{\mathtt{ym}} \cup \to_{\mathtt{ye_{AY}}} \cup \to_{\mathtt{ye_{YN}}}$.

▶ **Proposition 23.** *Let t be closed.*

1. *t is $\to_{\mathtt{w}\neg\mathtt{gcv}}$-normal if and only if t is $\to_{\mathtt{y}\neg\mathtt{gcv}}$-normal if and only if t is an answer.*
2. *t is $\to_{\mathtt{w}}$-normal if and only if t is $\to_{\mathtt{y}}$-normal if and only if t is a strict answer:*

$$\textsc{Strict answers} \quad a_s \quad ::= \quad v \mid a_s[x{\leftarrow}a'_s] \quad \text{with } x \in \mathtt{fv}(a_s)$$

Next, we prove that the CbS strategy is almost deterministic. Its non-erasing rules are deterministic, while $\to_{\mathtt{ygcv}}$ is diamond. For instance:
$\mathtt{I}[z{\leftarrow}\delta]\,{}_{\mathtt{ygcv}}{\leftarrow}\mathtt{I}[y{\leftarrow}\mathtt{I}][z{\leftarrow}\delta] \to_{\mathtt{ygcv}} \mathtt{I}[y{\leftarrow}\mathtt{I}]$, and then $\mathtt{I}[z{\leftarrow}\delta] \to_{\mathtt{ygcv}} \mathtt{I}\,{}_{\mathtt{ygcv}}{\leftarrow}\mathtt{I}[y{\leftarrow}\mathtt{I}]$.

▶ **Proposition 24.** *Reductions $\to_{\mathtt{ym}}$, $\to_{\mathtt{ye_{AY}}}$, and $\to_{\mathtt{ye_{YN}}}$ are deterministic, and $\to_{\mathtt{ygcv}}$ is diamond.*

Lastly, we prove the *linear* postponement of $\to_{\mathtt{ygcv}}$, that is, the fact that it can be postponed while preserving the length of the evaluation. The length preservation shall be used to prove the maximality of $\to_{\mathtt{y}}$ in the next section.

▶ **Proposition 25** (Postponement of $\to_{\mathtt{ygcv}}$)**.** *If $d : t \to_{\mathtt{y}}^{*} s$ then $t \to_{\mathtt{y}\neg\mathtt{gcv}}^{k} \to_{\mathtt{ygcv}}^{h} s$ with $k = |d|_{\mathtt{y}\neg\mathtt{gcv}}$ and $h = |d|_{\mathtt{ygcv}}$.*

9 Tight Derivations, Exact Lengths, and Maximality

Here, we focus on the CbS strategy on closed terms and isolate a class of *tight* type derivations whose indices measure exactly the number of non-erasing CbS steps, mimicking faithfully what was done in call-by-name/value/need by Accattoli et al. in [12]. An outcome shall be that the CbS strategy actually performs the *longest* weak evaluation on closed terms.

Tight Derivations. The basic tool for our quantitative analysis are tight type derivations, which shall be used to refine the correctness theorem of the weak case (Th. 11) in the case of closed terms. Tight derivations are defined via a predicate on their last judgement, as in [12].

▶ **Definition 26** (Tight types and derivations). *A type* T *is* tight *if* $T = \mathtt{norm}$ *or* $T = [\mathtt{norm}]$. *A derivation* $\pi \triangleright \Gamma \vdash^{(m,e)} t : T$ *is* tight *if* T *is tight.*

Tight Correctness. We first show that all tight derivations for answers have null indices.

▶ **Proposition 27** (Tight typing of normal forms for non-erasing CbS). *Let* a *be an answer and* $\pi \triangleright \Gamma \vdash^{(m,e)} a : \mathtt{norm}$ *be a derivation. Then* Γ *is empty and* $m = e = 0$.

Next, we refine subject reduction via tight derivations. The key point is that now the indices decrease of *exactly one* at each step. Tight correctness then follows.

▶ **Proposition 28** (Tight subject reduction for non-erasing CbS). *Let* $\pi \triangleright \Gamma \vdash^{(m,e)} t : \mathtt{norm}$ *be a tight derivation.*

1. Multiplicative: *if* $t \to_{\mathtt{ym}} s$ *then* $m \geq 1$ *and there is* $\rho \triangleright \Gamma \vdash^{(m-1,e)} s : \mathtt{norm}$.
2. Exponential: *if* $t \to_{\mathtt{ye_{AY}}} s$ *or* $t \to_{\mathtt{ye_{YN}}} s$ *then* $e \geq 1$ *and there is* $\rho \triangleright \Gamma \vdash^{(m,e-1)} s : \mathtt{norm}$.

▶ **Theorem 29** (Tight correctness for CbS). *Let* t *be a closed term and* $\pi \triangleright \vdash^{(m,e)} t : \mathtt{norm}$ *be a tight derivation. Then there is a weak normal form* n *such that* $d : t \to_{\mathtt{y}}^{*} n$ *with* $|d|_{\mathtt{ym}} = m$ *and* $|d|_{\mathtt{ye_{AY}},\mathtt{ye_{YN}}} = e$.

Proof. The proof is exactly as for weak correctness (Th. 11), except that if t is normal then the fact that $m = e = 0$ follows from Prop. 27 and that if t is not normal the equality on the number of steps is obtained by using tight subject reduction (Prop. 28) instead of the quantitative one. ◀

For tight completeness for closed terms, it is enough to observe that the statement of weak completeness (Th. 14) already gives a tight derivation $\Gamma \vdash t : \mathtt{norm}$, the type context Γ of which is empty because t is closed (Lemma 9).

▶ **Example 30.** We illustrate the tightness of multi types for CbS with an example. Consider the term $(\lambda y.yy)(\mathtt{II})$, the CbS evaluation of which is as follows (the first part coincides with the CbN evaluation):

CbN evaluation:

$(\lambda y.yy)(\mathtt{II})$

$\to_{\mathtt{ym}}$	$yy[y{\leftarrow}\mathtt{II}]$	$\to_{\mathtt{ye_{YN}}}$	$(\mathtt{II})y[y{\leftarrow}\mathtt{II}]$
$\to_{\mathtt{ym}}$	$(x[x{\leftarrow}\mathtt{I}])y[y{\leftarrow}\mathtt{II}]$	$\to_{\mathtt{ye_{YN}}}$	$(\mathtt{I}[x{\leftarrow}\mathtt{I}])y[y{\leftarrow}\mathtt{II}]$
$\to_{\mathtt{ym}}$	$z[z{\leftarrow}y][x{\leftarrow}\mathtt{I}][y{\leftarrow}\mathtt{II}]$	$\to_{\mathtt{ye_{YN}}}$	$y[z{\leftarrow}y][x{\leftarrow}\mathtt{I}][y{\leftarrow}\mathtt{II}]$
		$\to_{\mathtt{ye_{YN}}}$	$\mathtt{II}[z{\leftarrow}y][x{\leftarrow}\mathtt{I}][y{\leftarrow}\mathtt{II}]$
$\to_{\mathtt{ym}}$	$z'[z'{\leftarrow}\mathtt{I}][z{\leftarrow}y][x{\leftarrow}\mathtt{I}][y{\leftarrow}\mathtt{II}]$	$\to_{\mathtt{ye_{YN}}}$	$\mathtt{I}[z'{\leftarrow}\mathtt{I}][z{\leftarrow}y][x{\leftarrow}\mathtt{I}][y{\leftarrow}\mathtt{II}]$

CbS extension:

		$\to_{\mathtt{ye_{AY}}}$	$\mathtt{I}[z'{\leftarrow}\mathtt{I}][z{\leftarrow}\mathtt{II}][x{\leftarrow}\mathtt{I}][y{\leftarrow}\mathtt{II}]$
$\to_{\mathtt{ym}}$	$\mathtt{I}[z'{\leftarrow}\mathtt{I}][z{\leftarrow}z'[z'{\leftarrow}\mathtt{I}]][x{\leftarrow}\mathtt{I}][y{\leftarrow}\mathtt{II}]$	$\to_{\mathtt{ye_{YN}}}$	$\mathtt{I}[z'{\leftarrow}\mathtt{I}][z{\leftarrow}\mathtt{I}[z'{\leftarrow}\mathtt{I}]][x{\leftarrow}\mathtt{I}][y{\leftarrow}\mathtt{II}]$
$\to_{\mathtt{ym}}$	$\mathtt{I}[z'{\leftarrow}\mathtt{I}][z{\leftarrow}\mathtt{I}[z'{\leftarrow}\mathtt{I}]][x{\leftarrow}\mathtt{I}][y{\leftarrow}z'[z'{\leftarrow}\mathtt{I}]]$	$\to_{\mathtt{ye_{YN}}}$	$\mathtt{I}[z'{\leftarrow}\mathtt{I}][z{\leftarrow}\mathtt{I}[z'{\leftarrow}\mathtt{I}]][x{\leftarrow}\mathtt{I}][y{\leftarrow}\mathtt{I}[z'{\leftarrow}\mathtt{I}]]$

If we consider CbN evaluation, evaluation stops earlier: it would reach a normal form with $\mathtt{I}[z'{\leftarrow}\mathtt{I}][z{\leftarrow}y][x{\leftarrow}\mathtt{I}][y{\leftarrow}\mathtt{II}]$, that is after 4 multiplicative and 5 exponential steps. CbS evaluates more than CbN, hence the rewriting sequence ends only after 6 multiplicative and 8 exponential steps.

We now show a tight derivation for the term $(\lambda y.yy)(\mathtt{II})$, which is indeed indexed by $(6,8)$, as per tight correctness and completeness (Th. 29 and the text after it). For compactness, we shorten $\mathtt{norm}$ as $\mathtt{n}$. Moreover, to fit the derivation in the page, we first give the main derivation, giving a name to some sub-derivations which are shown after the main one.

- Main derivation:

$$\dfrac{\pi \vdash^{(1,3)} \lambda y.yy : [[\mathtt{n}]\to\mathtt{n},\mathtt{n},\mathtt{n}]\to\mathtt{n} \qquad \dfrac{\rho \vdash^{(1,2)} \mathtt{II} : [\mathtt{n}]\to\mathtt{n} \qquad (\rho_{\mathtt{n}} \vdash^{(1,1)} \mathtt{II} : \mathtt{n}) \times 3}{\vdash^{(4,5)} \mathtt{II} : [[\mathtt{n}]\to\mathtt{n},\mathtt{n},\mathtt{n}] \uplus [\mathtt{n}]}\,\mathsf{many}}{\vdash^{(6,8)} (\lambda y.yy)(\mathtt{II}) : \mathtt{n}}\,@$$

- Auxiliary ones:

$$\pi := \dfrac{\dfrac{y : [[\mathtt{n}]\to\mathtt{n}] \vdash^{(0,1)} y : [\mathtt{n}]\to\mathtt{n} \qquad \dfrac{y : [\mathtt{n}] \vdash^{(0,1)} y : \mathtt{n} \qquad y : [\mathtt{n}] \vdash^{(0,1)} y : \mathtt{n}}{y : [\mathtt{n},\mathtt{n}] \vdash^{(0,2)} y : [\mathtt{n}] \uplus [\mathtt{n}]}\,\mathsf{many}}{y : [[\mathtt{n}]\to\mathtt{n},\mathtt{n},\mathtt{n}] \vdash^{(1,3)} yy : \mathtt{n}}\,@}{\vdash^{(1,3)} \lambda y.yy : [[\mathtt{n}]\to\mathtt{n},\mathtt{n},\mathtt{n}]\to\mathtt{n}}\,\lambda$$

$$\rho_{\mathtt{n}} := \dfrac{\pi_{\mathtt{n}} \vdash^{(0,1)} \mathtt{I} : [[\mathtt{n}]\to[\mathtt{n}] \qquad \dfrac{\dfrac{}{\vdash^{(0,0)} \mathtt{I} : \mathtt{n}}\,\mathsf{ax}_\lambda \qquad \dfrac{}{\vdash^{(0,0)} \mathtt{I} : \mathtt{n}}\,\mathsf{ax}_\lambda}{\vdash^{(0,0)} \mathtt{I} : [\mathtt{n}] \uplus [\mathtt{n}]}\,\mathsf{many}}{\vdash^{(1,1)} \mathtt{II} : \mathtt{n}}\,@$$

$$\rho := \dfrac{\dfrac{\dfrac{}{z : [[\mathtt{n}]\to\mathtt{n}] \vdash^{(0,1)} z : [\mathtt{n}]\to\mathtt{n}}\,\mathsf{ax}}{\vdash^{(0,1)} \mathtt{I} : [[\mathtt{n}]\to\mathtt{n}]\to([\mathtt{n}]\to\mathtt{n})}\,\lambda \qquad \dfrac{\pi_{\mathtt{n}} \vdash^{(0,1)} \mathtt{I} : [\mathtt{n}]\to\mathtt{n} \qquad \dfrac{}{\vdash^{(0,0)} \mathtt{I} : \mathtt{n}}\,\mathsf{ax}_\lambda}{\vdash^{(0,1)} \mathtt{I} : [[\mathtt{n}]\to\mathtt{n}] \uplus [\mathtt{n}]}\,\mathsf{many}}{\vdash^{(1,2)} \mathtt{II} : [\mathtt{n}]\to\mathtt{n}}\,@$$

Maximality of the CbS Strategy. Similarly to how we proved uniform normalization for $\to_{\mathtt{w}}$, we can prove that on closed terms the CbS strategy does reach a weak normal form whenever one exists – the key point being that $\to_{\mathtt{y}}$ does not stop too soon. This fact proves that $\simeq_C^{\mathtt{silly}}$ can equivalently be defined using $\to_{\mathtt{y}}$, as mentioned in Sect. 7. Moreover, by exploiting tight correctness and some of the rewriting properties of Sect. 4, we prove that the CbS strategy is maximal.

▶ **Proposition 31.** *Let t be closed and $t \to_{\mathtt{w}}^{h} a_s$ with a_s a strict answer.*

1. CbS is normalizing*: $t \to_{\mathtt{y}}^{k} a_s$ for some $k \in \mathbb{N}$;*
2. CbS is maximal*: $h \leq k$.*

Proof.

1. By completeness (Th. 14), $t \to_{\mathtt{w}}^{h} a_s$ implies typability of t (a strict answer is in particular a weak normal form), which in turn, by tight correctness (Th. 29), implies $t \to_{\mathtt{y}}^{k} a'_s$ for some $k \in \mathbb{N}$. By confluence (Th. 7), $a_s = a'_s$.
2. By postponement of $\to_{\mathtt{wgcv}}$ (Prop. 3) applied to $t \to_{\mathtt{w}}^{h} a_s$, we obtain a sequence $d : t \to_{\mathtt{w}\neg\mathtt{gcv}}^{h_1} s \to_{\mathtt{wgcv}}^{h_2} a_s$ with $h_1 + h_2 \geq h$, for some s. For the strategy sequence $t \to_{\mathtt{y}}^{k} a_s$ given by Point 1, the *moreover* part of the postponement property gives us a sequence $e : t \to_{\mathtt{y}\neg\mathtt{gcv}}^{k_1} u \to_{\mathtt{ygcv}}^{k_2} a_s$ with, crucially, $k_1 + k_2 = k$, for some u. By completeness of the type system (Th. 14), we obtain a derivation $\vdash^{(m,e)} t : \mathtt{norm}$. By weak correctness (Th. 11), we obtain $h_1 \leq m + e$. By tight correctness (Th. 29), $k_1 = m + e$. Thus, $h_1 \leq k_1$.

Now, note that both s and u are $\to_{\mathtt{w}\neg\mathtt{gcv}}$-normal, because otherwise, by (iterated) strict commutation of $\to_{\mathtt{w}\neg\mathtt{gcv}}$ and $\to_{\mathtt{wgcv}}$ (Lemma 6), we obtain that a_s is not $\to_{\mathtt{w}}$-normal, against hypothesis. Since $\to_{\mathtt{w}\neg\mathtt{gcv}}$ is confluent (Th. 7), $s = u$. Since $\to_{\mathtt{wgcv}}$ is diamond (Lemma 5), $h_2 = k_2$. Then $k = k_1 + k_2 = k_1 + h_2 \geq h_1 + h_2 = h$. ◀

Why not the λ-Calculus? In the λ-calculus, it is hard to specify via evaluation contexts the idea behind the CbS strategy of *evaluating arguments only when they are no longer needed*. The difficulty is specific to weak evaluation. For instance, for $t := (\lambda y.\lambda x.yy)u$ the CbS strategy should evaluate u before substituting it for y, because once u ends up under λx it shall be unreachable by weak evaluation. For $s := (\lambda y.(\lambda x.yy)r)u$, instead, the CbS strategy should not evaluate u before substituting it, because weak evaluation will reach the two occurrences of y, and one obtains a longer reduction by substituting u before evaluating it. Note that one cannot decide what to do with u in t and s by checking if y occurs inside the abstraction, because y occurs under λy in both t and s.

Multi types naturally make the right choices for t (evaluating u before substituting it) and s (substituting u before evaluating it), so a strategy matching exactly the bounds given by multi types needs to do the same choices. The LSC allows one to bypass the difficulty, by first turning β-redexes into explicit substitutions, thus exposing y out of abstractions in s but not in t, and matching what is measured by multi types. In the λ-calculus, we have not found natural ways of capturing this aspect (be careful: the example illustrates the problem but solving the problem requires more than just handling the example).

10 Conclusions

We introduce the weak silly substitution calculus and the CbS strategy by mirroring the properties of CbNeed with respect to duplication and erasure. Then, we provide evidence of the good design of the framework via a rewriting study of the calculus and the strategy, and by mirroring the semantic analyses of CbNeed via multi types by Kesner [38] (qualitative) and Accattoli et al. [12] (quantitative).

Conceptually, the main results are the operational equivalence of CbS and CbV, mirroring the one between CbN and CbNeed, and the exact measuring of CbS evaluation lengths via multi types, having the interesting corollary that CbS is *maximal* in the Weak SSC. It would be interesting to show that, dually, the CbNeed strategy computes the shortest reduction in the CbNeed LSC. We are not aware of any such result.

Our work also shows that CbV contextual equivalence $\simeq_C^{\mathtt{value}}$ is completely blind to the efficiency of evaluation. We think that it is important to look for natural refinements of $\simeq_C^{\mathtt{value}}$ which are less blind, or, in the opposite direction, exploring what are the minimal extensions of CbV that make $\simeq_C^{\mathtt{value}}$ efficiency-sensitive.

We would also like to develop categorical and game semantics for both CbS and CbNeed, to pinpoint the principles behind the wiseness and the silliness of duplications and erasures.

References

1 Beniamino Accattoli. An abstract factorization theorem for explicit substitutions. In Ashish Tiwari, editor, *23rd International Conference on Rewriting Techniques and Applications (RTA'12) , RTA 2012, May 28 - June 2, 2012, Nagoya, Japan*, volume 15 of *LIPIcs*, pages 6–21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.RTA.2012.6.

2 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.

3 Beniamino Accattoli. Proof nets and the linear substitution calculus. In Bernd Fischer and Tarmo Uustalu, editors, *Theoretical Aspects of Computing - ICTAC 2018 - 15th International Colloquium, Stellenbosch, South Africa, October 16-19, 2018, Proceedings*, volume 11187 of *Lecture Notes in Computer Science*, pages 37–61. Springer, 2018. doi:10.1007/978-3-030-02508-3_3.

4 Beniamino Accattoli. Exponentials as substitutions and the cost of cut elimination in linear logic. In Christel Baier and Dana Fisman, editors, *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2 - 5, 2022*, pages 49:1–49:15. ACM, 2022. doi:10.1145/3531130.3532445.

5 Beniamino Accattoli, Pablo Barenbaum, and Damiano Mazza. Distilling abstract machines. In Johan Jeuring and Manuel M. T. Chakravarty, editors, *Proceedings of the 19th ACM SIGPLAN international conference on Functional programming, Gothenburg, Sweden, September 1-3, 2014*, pages 363–376. ACM, 2014. doi:10.1145/2628136.2628154.

6 Beniamino Accattoli and Bruno Barras. Environments and the complexity of abstract machines. In *Proceedings of the 19th International Symposium on Principles and Practice of Declarative Programming (PPDP 2017)*, pages 4–16, 2017. doi:10.1145/3131851.3131855.

7 Beniamino Accattoli and Bruno Barras. The negligible and yet subtle cost of pattern matching. In Bor-Yuh Evan Chang, editor, *Programming Languages and Systems - 15th Asian Symposium, APLAS 2017, Suzhou, China, November 27-29, 2017, Proceedings*, volume 10695 of *Lecture Notes in Computer Science*, pages 426–447. Springer, 2017. doi:10.1007/978-3-319-71237-6_21.

8 Beniamino Accattoli, Eduardo Bonelli, Delia Kesner, and Carlos Lombardi. A nonstandard standardization theorem. In Suresh Jagannathan and Peter Sewell, editors, *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 659–670. ACM, 2014. doi:10.1145/2535838.2535886.

9 Beniamino Accattoli, Stéphane Graham-Lengrand, and Delia Kesner. Tight typings and split bounds, fully developed. *J. Funct. Program.*, 30:e14, 2020. doi:10.1017/S095679682000012X.

10 Beniamino Accattoli and Giulio Guerrieri. Open call-by-value. In Atsushi Igarashi, editor, *Programming Languages and Systems - 14th Asian Symposium, APLAS 2016, Hanoi, Vietnam, November 21-23, 2016, Proceedings*, volume 10017 of *Lecture Notes in Computer Science*, pages 206–226, 2016. doi:10.1007/978-3-319-47958-3_12.

11 Beniamino Accattoli and Giulio Guerrieri. The theory of call-by-value solvability. *Proc. ACM Program. Lang.*, 6(ICFP):855–885, 2022. doi:10.1145/3547652.

12 Beniamino Accattoli, Giulio Guerrieri, and Maico Leberle. Types by need. In *Proceedings of the 28th European Symposium on Programming (ESOP 2019)*, pages 410–439, 2019. doi:10.1007/978-3-030-17184-1_15.

13 Beniamino Accattoli and Delia Kesner. The structural *lambda*-calculus. In Anuj Dawar and Helmut Veith, editors, *Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23-27, 2010. Proceedings*, volume 6247 of *Lecture Notes in Computer Science*, pages 381–395. Springer, 2010. doi:10.1007/978-3-642-15205-4_30.

14 Beniamino Accattoli and Adrienne Lancelot. Mirroring call-by-need, or values acting silly, 2024. arXiv:2402.12078.

15 Beniamino Accattoli and Maico Leberle. Useful open call-by-need. In Florin Manea and Alex Simpson, editors, *30th EACSL Annual Conference on Computer Science Logic, CSL 2022, February 14-19, 2022, Göttingen, Germany (Virtual Conference)*, volume 216 of *LIPIcs*, pages 4:1–4:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.CSL.2022.4.

16 Beniamino Accattoli and Luca Paolini. Call-by-value solvability, revisited. In Tom Schrijvers and Peter Thiemann, editors, *Functional and Logic Programming - 11th International Symposium, FLOPS 2012, Kobe, Japan, May 23-25, 2012. Proceedings*, volume 7294 of *Lecture Notes in Computer Science*, pages 4–16. Springer, 2012. doi:10.1007/978-3-642-29822-6_4.

17 Zena M. Ariola, Paul Downen, Hugo Herbelin, Keiko Nakata, and Alexis Saurin. Classical call-by-need sequent calculi: The unity of semantic artifacts. In Tom Schrijvers and Peter Thiemann, editors, *Functional and Logic Programming - 11th International Symposium, FLOPS 2012, Kobe, Japan, May 23-25, 2012. Proceedings*, volume 7294 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 2012. doi:10.1007/978-3-642-29822-6_6.

18 Zena M. Ariola, Matthias Felleisen, John Maraist, Martin Odersky, and Philip Wadler. The call-by-need lambda calculus. In Ron K. Cytron and Peter Lee, editors, *Conference Record of POPL'95: 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Francisco, California, USA, January 23-25, 1995*, pages 233–246. ACM Press, 1995. doi:10.1145/199448.199507.

19 Zena M. Ariola, Hugo Herbelin, and Alexis Saurin. Classical call-by-need and duality. In C.-H. Luke Ong, editor, *Typed Lambda Calculi and Applications - 10th International Conference, TLCA 2011, Novi Sad, Serbia, June 1-3, 2011. Proceedings*, volume 6690 of *Lecture Notes in Computer Science*, pages 27–44. Springer, 2011. doi:10.1007/978-3-642-21691-6_6.

20 Pablo Arrighi and Gilles Dowek. Lineal: A linear-algebraic lambda-calculus. *Log. Methods Comput. Sci.*, 13(1), 2017. doi:10.23638/LMCS-13(1:8)2017.

21 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.

22 Thibaut Balabonski, Antoine Lanco, and Guillaume Melquiond. A strong call-by-need calculus. In Naoki Kobayashi, editor, *6th International Conference on Formal Structures for Computation and Deduction, FSCD 2021, July 17-24, 2021, Buenos Aires, Argentina (Virtual Conference)*, volume 195 of *LIPIcs*, pages 9:1–9:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.FSCD.2021.9.

23 Pablo Barenbaum, Eduardo Bonelli, and Kareem Mohamed. Pattern matching and fixed points: Resource types and strong call-by-need: Extended abstract. In David Sabel and Peter Thiemann, editors, *Proceedings of the 20th International Symposium on Principles and Practice of Declarative Programming, PPDP 2018, Frankfurt am Main, Germany, September 03-05, 2018*, pages 6:1–6:12. ACM, 2018. doi:10.1145/3236950.3236972.

24 Alexis Bernadet and Stéphane Lengrand. Complexity of strongly normalising λ-terms via non-idempotent intersection types. In *FOSSACS 2011*, pages 88–107, 2011. doi:10.1007/978-3-642-19805-2_7.

25 Dariusz Biernacki, Sergueï Lenglet, and Piotr Polesiuk. A complete normal-form bisimilarity for state. In Mikolaj Bojanczyk and Alex Simpson, editors, *Foundations of Software Science and Computation Structures - 22nd International Conference, FOSSACS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings*, volume 11425 of *Lecture Notes in Computer Science*, pages 98–114. Springer, 2019. doi:10.1007/978-3-030-17127-8_6.

26 Antonio Bucciarelli, Delia Kesner, and Simona Ronchi Della Rocca. Solvability = typability + inhabitation. *Log. Methods Comput. Sci.*, 17(1), 2021. URL: https://lmcs.episciences.org/7141.

27 Alberto Carraro and Giulio Guerrieri. A semantical and operational account of call-by-value solvability. In Anca Muscholl, editor, *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*, volume 8412 of *Lecture Notes in Computer Science*, pages 103–118. Springer, 2014. doi:10.1007/978-3-642-54830-7_7.

28 Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In Martin Odersky and Philip Wadler, editors, *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00), Montreal, Canada, September 18-21, 2000*, pages 233–243. ACM, 2000. doi:10.1145/351240.351262.

29 Ugo Dal Lago and Simone Martini. The weak lambda calculus as a reasonable machine. *Theor. Comput. Sci.*, 398(1-3):32–50, 2008. doi:10.1016/j.tcs.2008.01.044.

30 Daniel de Carvalho. *Sémantiques de la logique linéaire et temps de calcul*. Thèse de doctorat, Université Aix-Marseille II, 2007.

31 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.

32 Thomas Ehrhard. Collapsing non-idempotent intersection types. In Patrick Cégielski and Arnaud Durand, editors, *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012, September 3-6, 2012, Fontainebleau, France*, volume 16 of *LIPIcs*, pages 259–273. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.CSL.2012.259.

33 Thomas Ehrhard and Giulio Guerrieri. The bang calculus: an untyped lambda-calculus generalizing call-by-name and call-by-value. In James Cheney and Germán Vidal, editors, *Proceedings of the 18th International Symposium on Principles and Practice of Declarative Programming, Edinburgh, United Kingdom, September 5-7, 2016*, pages 174–187. ACM, 2016. doi:10.1145/2967973.2968608.

34 Claudia Faggian and Simona Ronchi Della Rocca. Lambda calculus and probabilistic computation. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, pages 1–13. IEEE, 2019. doi:10.1109/LICS.2019.8785699.

35 Philippa Gardner. Discovering needed reductions using type theory. In Masami Hagiya and John C. Mitchell, editors, *Theoretical Aspects of Computer Software, International Conference TACS '94, Sendai, Japan, April 19-22, 1994, Proceedings*, volume 789 of *Lecture Notes in Computer Science*, pages 555–574. Springer, 1994. doi:10.1007/3-540-57887-0_115.

36 J.R. Hindley. *The Church-Rosser Property and a Result in Combinatory Logic*. PhD thesis, University of Newcastle-upon-Tyne, 1964.

37 Axel Kerinec, Giulio Manzonetto, and Simona Ronchi Della Rocca. Call-by-value, again! In Naoki Kobayashi, editor, *6th International Conference on Formal Structures for Computation and Deduction, FSCD 2021, July 17-24, 2021, Buenos Aires, Argentina (Virtual Conference)*, volume 195 of *LIPIcs*, pages 7:1–7:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.FSCD.2021.7.

38 Delia Kesner. Reasoning about call-by-need by means of types. In Bart Jacobs and Christof Löding, editors, *Foundations of Software Science and Computation Structures - 19th International Conference, FOSSACS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9634 of *Lecture Notes in Computer Science*, pages 424–441. Springer, 2016. doi:10.1007/978-3-662-49630-5_25.

39 Delia Kesner and Shane Ó Conchúir. Milner's lambda calculus with partial substitutions. Technical report, Paris 7 University, 2008. URL: http://www.pps.univ-paris-diderot.fr/~kesner/papers/shortpartial.pdf.

40 Delia Kesner, Loïc Peyrot, and Daniel Ventura. The spirit of node replication. In Stefan Kiefer and Christine Tasson, editors, *Foundations of Software Science and Computation Structures - 24th International Conference, FOSSACS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings*, volume 12650 of *Lecture Notes in Computer Science*, pages 344–364. Springer, 2021. doi:10.1007/978-3-030-71995-1_18.

41 Delia Kesner, Alejandro Ríos, and Andrés Viso. Call-by-need, neededness and all that. In Christel Baier and Ugo Dal Lago, editors, *Foundations of Software Science and Computation Structures - 21st International Conference, FOSSACS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*, volume 10803 of *Lecture Notes in Computer Science*, pages 241–257. Springer, 2018. doi:10.1007/978-3-319-89366-2_13.

42 Delia Kesner and Daniel Ventura. Quantitative types for the linear substitution calculus. In Josep Díaz, Ivan Lanese, and Davide Sangiorgi, editors, *Theoretical Computer Science - 8th IFIP TC 1/WG 2.2 International Conference, TCS 2014, Rome, Italy, September 1-3, 2014. Proceedings*, volume 8705 of *Lecture Notes in Computer Science*, pages 296–310. Springer, 2014. doi:10.1007/978-3-662-44602-7_23.

43 A. J. Kfoury. A linearization of the lambda-calculus and consequences. *J. Log. Comput.*, 10(3):411–436, 2000. doi:10.1093/LOGCOM/10.3.411.

44 Olivier Laurent. *Étude de la polarisation en logique*. Thèse de doctorat, Université Aix-Marseille II, March 2002.

45 Paul Blain Levy. Call-by-push-value: Decomposing call-by-value and call-by-name. *High. Order Symb. Comput.*, 19(4):377–414, 2006. doi:10.1007/s10990-006-0480-6.

46 Giulio Manzonetto, Michele Pagani, and Simona Ronchi Della Rocca. New semantical insights into call-by-value λ-calculus. *Fundam. Informaticae*, 170(1-3):241–265, 2019. doi:10.3233/FI-2019-1862.

47 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.

48 Robin Milner. Local bigraphs and confluence: Two conjectures: (extended abstract). In Roberto M. Amadio and Iain Phillips, editors, *Proceedings of the 13th International Workshop on Expressiveness in Concurrency, EXPRESS 2006, Bonn, Germany, August 26, 2006*, volume 175 of *Electronic Notes in Theoretical Computer Science*, pages 65–73. Elsevier, 2006. doi:10.1016/j.entcs.2006.07.035.

49 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.

50 György E. Révész. A list-oriented extension of the lambda-calculus satisfying the Church-Rosser theorem. *Theor. Comput. Sci.*, 93(1):75–89, 1992. doi:10.1016/0304-3975(92)90212-X.

51 Alexis Saurin. On the relations between the syntactic theories of lambda-mu-calculi. In Michael Kaminski and Simone Martini, editors, *Computer Science Logic, 22nd International Workshop, CSL 2008, 17th Annual Conference of the EACSL, Bertinoro, Italy, September 16-19, 2008. Proceedings*, volume 5213 of *Lecture Notes in Computer Science*, pages 154–168. Springer, 2008. doi:10.1007/978-3-540-87531-4_13.

52 Kristian Støvring and Søren B. Lassen. A complete, co-inductive syntactic theory of sequential control and state. In Jens Palsberg, editor, *Semantics and Algebraic Specification, Essays Dedicated to Peter D. Mosses on the Occasion of His 60th Birthday*, volume 5700 of *Lecture Notes in Computer Science*, pages 329–375. Springer, 2009. doi:10.1007/978-3-642-04164-8_17.

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

IMELL Cut Elimination with Linear Overhead

Beniamino Accattoli
Inria & LIX, École Poytechnique, UMR 7161, Palaiseau, France

Claudio Sacerdoti Coen
Alma Mater Studiorum – Università di Bologna, Italy

Abstract

Recently, Accattoli introduced the Exponential Substitution Calculus (ESC) given by untyped proof terms for Intuitionistic Multiplicative Exponential Linear Logic (IMELL), endowed with rewriting rules at-a-distance for cut elimination. He also introduced a new cut elimination strategy, dubbed *the good strategy*, and showed that its number of steps is a time cost model with polynomial overhead for ESC/IMELL, and the first such one.

Here, we refine Accattoli's result by introducing an abstract machine for ESC and proving that it implements the good strategy and computes cut-free terms/proofs within a *linear* overhead.

2012 ACM Subject Classification Theory of computation → Operational semantics

Keywords and phrases Lambda calculus, linear logic, abstract machines

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.24

Related Version *Extended Version*: `https://arxiv.org/abs/2405.03669` [11]

Supplementary Material *Software (Source Code)*: `https://github.com/sacerdot/sesame/` [41] archived at `swh:1:dir:93776e14435d0dd5cde6deee3d9cf8f515f18fac`

Funding *Claudio Sacerdoti Coen*: Supported by the INdAM-GNCS project "Modelli Composizionali per l'Analisi di Sistemi Reversibili Distribuiti" and by the Cost Action CA2011 EuroProofNet.

1 Introduction

One of the advances of the last decade in the theory of λ-calculus is the study of reasonable cost models carried out by Accattoli, Dal Lago, and co-authors [14, 15, 10, 17, 19], completing the research program started by Dal Lago and Martini almost 20 years ago [27, 34, 29, 28].

The General Problem. Focusing on time, the underlying problem is whether the number of β-steps (of a fixed evaluation strategy) can be taken as a measure of time that is polynomially equivalent to that of random access machines (RAMs); or, equivalently, to that of Turing machines. This is a priori unclear because there are families of terms suffering of *size explosion*: the size of terms grows exponentially with the number of β-steps, independently of the adopted evaluation strategy. The difficulty is finding a strategy, together with a simulation on RAMs of the strategy, working within a overhead polynomial in:

- *Length*: the number of steps of the strategy, and
- *Input*: the size of the initial term.

When a strategy admits such a simulation we say that it is a *a polynomial cost model*. Note that the number of steps of the strategy can be whatever (it need not be polynomial), it is the *overhead* of the simulation that has to be polynomial in the above parameters.

The overhead is required to be polynomial because reasonable frameworks are not necessarily linearly related; this is the reason why the complexity class P is so important. For instance, Turing machines simulate RAMs only within a quadratic overhead. When possible, however, it is interesting to know what is the exact degree of the overhead with

respect to RAMs, which are the commonly accepted abstraction of modern computers. When the overhead is linear – which is the optimal situation – then the polynomial cost model is trustable in practice: it does not hide an acceptable but possibly costly polynomial overhead.

Today, it is well-known that strategies such as weak head, head, and leftmost β-reduction provide polynomial time cost models, and that the same is true for their call-by-value and call-by-need variants [24, 42, 34, 14, 15, 10, 17, 22, 20, 23].

LSC and the Sub-Term Property. The key tool for the polynomially bounded simulations underlying the recent advances is Accattoli and Kesner's *linear substitution calculus* (shortened to LSC) [2, 9], which is an intermediary setting between the λ-calculus and RAMs.

The LSC is a simple λ-calculus with explicit substitutions refining a previous calculus by Milner [40, 32] and where evaluation is *micro-step*, that is, substitution acts on a variable occurrence at a time, rather than on all at once (which is *small-step*). A distinguished feature of the LSC is its rewriting rules *at a distance*, that is, rules where explicit substitutions do not percolate through the term structure but rather act through some contexts. The relevance of the LSC for reasonable time is linked to the *sub-term property* of its standard strategies [15]: at any point, only sub-terms of the initial term are duplicated, thus allowing one to bound the cost of each duplication with the size of the input.

No strategy of the λ-calculus has the sub-term property, which, as discussed by Accattoli [4], is the essence of the size explosion problem. The LSC crystallizes exactly what is needed for refining the λ-calculus to retrieve the sub-term property and circumvent size explosion.

LSC and Abstract Machines. It is also well-known that strategies of the LSC can usually be implemented with environment-based abstract machines whose overhead is *bi-linear*, that is, linear in both the input and length parameters, as shown by Accattoli et al. [6]. Abstract machines handle three tasks, namely the *decomposition of the substitution process*, the *search for redexes*, and α*-renaming*. The difference between the LSC and abstract machines is that the LSC only handles the substitution process, the critical one for avoiding size explosion. Search and α-renaming, indeed, tend to take only a linear overhead. That said, handling search and α-renaming provides an in-depth understanding of evaluation and it is mandatory for obtaining precise bounds from the cost model.

Lifting to Linear Logic. In 2022, Accattoli started a generalization of the mentioned recent results for the λ-calculus to the wider framework of linear logic [4]. Linear logic can be seen as a micro-step system with a tight control over duplications, somewhat similar to the LSC. Accattoli's starting point is the observation that, despite such similarity, no reasonable cut elimination strategy for linear logic was available. In particular, no cut elimination strategy with the sub-term property was known.

In [4], he introduces a generalization of the LSC to intuitionistic multiplicative exponential linear logic (IMELL). His *exponential substitution calculus* (ESC) is an untyped calculus of proof terms for the sequent calculus proofs of IMELL, endowed with cut elimination at a distance and having IMELL as typing system. He then designs a new cut elimination strategy, dubbed *good strategy*, which has the sub-term property and which is a polynomial cost model for ESC, the first such cost model for an expressive fragment of linear logic.

This Paper. Here, we design an environment-based abstract machine implementing the good strategy of ESC, show that it has bi-linear overhead, and provide an OCaml implementation. The result is not surprising, and yet we believe that it is interesting, for various reasons.

Firstly, it is the first linear overhead result for an expressive fragment of linear logic. It allows one to extract precise time bounds from the length of cut elimination, and can thus be useful, for instance, in linear-logic-based implicit computational complexity.

Secondly, our study differs in many details from other environment-based machines. There are in fact at least two literatures related to our work (surveyed in the next paragraph). A recent one about cost analyses of machines for λ-calculi and an older one about machines for linear logic, here referred to as *linear machines*. The main differences are the following:

- *Natural deduction vs sequent calculus*: machines usually implement calculi based on natural deduction, while ESC is based on sequent calculus, and some aspects are different. For instance, our machine has no argument stack (as it is expected, after Herbelin [31]).
- *Micro-step source and no structural equivalence*: the implemented calculus is usually small-step, while here it is micro-step. In the literature, when the calculus is micro-step it is usually implemented by the machine up to a notion of structural equivalence. Here, there is no need of structural equivalence.
- *Strong vs weak*: our machine computes cut-free proofs, while other linear machines in the literature do not, as they usually only perform weak (or surface) evaluation (that is, they do not evaluate under abstraction and inside promotions). To our knowledge, ours is the first strong linear machine.
- *Strong with no backtracking*: machines for strong evaluation usually have transitions for sequentially backtracking once the evaluation of a sub-term is over. Here, we adopt a recent new technique for strong machines by Accattoli and Barenbaum [5], that avoids sequential backtracking. The idea is to assign distinct jobs to each sub-term and simply jumping to the next job once the current one is over. This approach structures the machine in a very different way and provides drastically simpler strong machines.
- *Complexity analysis*: the complexity of linear machines is never studied in the literature.
- *OCaml implementation*: linear machines are usually studied theoretically and never implemented. We provide an OCaml implementation verifying our complexity analysis. In particular, it is the first implementation of the technique for strong machines of [5].

Summing up, our machine is simple, given the complex setting that it implements, thanks to the absence of stacks and backtracking. In fact, it uses just one (non-trivial) data structure.

Methodologically, the main difference between our work and the literature on linear machines is that those machines were developed to provide insights about λ-calculi and functional programming (such as no garbage collection, in-place updates, and single pointer property) while here we proceed the other way around, using the recent theory of the λ-calculus to provide insights about linear logic.

Architecture of the Result. The good strategy that we implement is a non-deterministic but *diamond* strategy, where diamond means that the choices do not affect termination nor evaluation lengths (otherwise the number of steps would not be a well-defined cost model). Our implementation is in two phases. The first and main one, performed by a deterministic machine dubbed *SESAME* (Strong Exponential Substitution Abstract Machine without Erasure), never erases. The second one is a simple garbage collection pass over the output of SESAME, that produces a cut-free term. For the correctness of SESAME, we thus need to relate a non-deterministic strategy and a deterministic machine, which is slightly unusual. For that, we follow the abstract recipe of Accattoli et al. [12] which is here simplified because our case is not up to structural equivalence and there is no implosiveness.

Related Work. Accattoli and co-authors studied at length the overhead of machines for λ-calculi both micro-step [6, 7, 10] and small-step [3, 8, 18, 13, 12, 19, 5]. Biernacka et al. similarly studied machines for strong call-by-value [22] and strong call-by-need [23].

Lafont considered the first linear logic (LL) machine, based on categorical combinators for intuitionistic LL (ILL) [33]. Abramsky considered an environment-based machine for ILL and a chemical-style machine for LL, both doing only weak evaluation [1]. The latter machine was then studied by Mikami and Akama [39] and Sato and co-authors [43, 38]. Turner and Wadler study the use of IMELL for memory management and give two machines, one with the single pointer property and one without it but with a memoization mechanism [45]. Alberti and Ritter also deal with the single pointer property [21]. Bonelli gives a machine based on an unusual sequent calculus for ILL [25].

The interaction abstract machine (IAM) is an unusual token-based machine for LL proofs first studied by Danos and Regnier [30] and Mackie [35]. Recent work by Accattoli et al. showed that the IAM overhead is unreasonable for both time and space [16, 17, 46]. Mackie also studied interaction nets-based implementations of LL [36, 37].

2 The Exponential Substitution Calculus

In this section, we briefly recall Accattoli's exponential substitution calculus (ESC) [4] which is an untyped calculus having IMELL as system of simple types.

For more explanations, we refer the reader to [4]. The main differences with respect to [4] is that, for lack of space, we omit tensor and we only deal with micro-step rewriting rules (omitting the small-step exponential rule, which is derivable). In IMELL, linear implication is "more important" than tensor, as tensors are not needed to simulate the λ-calculus. In Appendix B of the technical report [11], we explain how to straightforwardly extend our study to tensor.

Values and Terms. The grammars of ESC are in the upper part of Fig. 1. Variables are of two disjoint kinds, multiplicative and exponential, and we refer to variables of unspecified kind using x, y, z. Values are the proof terms associated to axioms or to the right rules and, beyond variables, are *abstractions* $\lambda x.t$ and *promotions* $!t$. The proof terms decorating left rules are *subtractions* $[m \otimes v, x]t$, *derelictions* $[e?x]t$, and *cuts* $[v \to x]t$, which is in red because of its special role. Note that cuts and subtractions are *split*, that is, have values (rather than terms) as left sub-terms. The constructors $\lambda x.t$, $[m \otimes v, x]t$, $[e?x]t$, and $[v \to x]t$ bind x in t. We identify terms up to α-renaming. Free variables (resp. multiplicative/exponential variables) are defined as expected, and noted $\mathtt{fv}(t)$ (resp. $\mathtt{mfv}(t)$ and $\mathtt{efv}(t)$). We use $|t|$ for the number of constructors in t, and $|t|_x$ for the number of free occurrences of x in t. There is a notion of *proper term* ensuring the linearity of multiplicative variables and the exponential boundary of promotions, detailed in Appendix A of the tech report [11]. The only relevant case is: $!t$ *is proper if* t *is proper and* $\mathtt{mfv}(t) = \emptyset$. In the following, terms are assumed to be proper.

Contexts and Plugging. The broadest notion of context that we consider is *general contexts* C, which simply allow the hole $\langle\cdot\rangle$ to replace any sub-term in a term. Because of split cuts and subtractions (that is, the fact that their left sub-term is a value rather than a term), the definition relies on the auxiliary notion of *value context* V. The definition also uses *left contexts* L, which are contexts under left constructors (or, for binary left constructors, under the right sub-term) that play a key role in the system – their use in defining C is just to keep the grammar compact. We also need *cut contexts* E, which are noted with E because they shall play the role of machine environments in Sect. 5.

Mult. values	$v_{\mathtt{m}}$	$::=$	$m \mid \lambda x.t$	Values	v, v'	$::=$	$v_{\mathtt{m}} \mid v_{\mathtt{e}}$
Exp. values	$v_{\mathtt{e}}$	$::=$	$e \mid !t$	Terms	t	$::=$	$v \mid [v{\to}x]t \mid [m{\otimes}v, x]t \mid [e?x]t$

Cut ctxs	$E ::= \langle\cdot\rangle \mid [v{\to}x]E$	Left ctxs	$L ::= \langle\cdot\rangle \mid [v{\to}x]L \mid [m{\otimes}v, x]L \mid [e?x]L$
Value ctxs	$V ::= \langle\cdot\rangle \mid \lambda x.C \mid !C$	Ctxs	$C ::= V \mid [V{\to}x]t \mid [m{\otimes}V, x]t \mid L\langle C\rangle$

Micro-step root multiplicative rules

$$
\begin{array}{rcll}
[v_{\mathtt{m}}{\to}m]C\langle\!\langle m\rangle\!\rangle & \mapsto_{\mathtt{axm}_1} & C\langle\!\langle v_{\mathtt{m}}\rangle\!\rangle & \\
[n{\to}m]C\langle[m{\otimes}v, x]t\rangle & \mapsto_{\mathtt{axm}_2} & C\langle[n{\otimes}v, x]t\rangle & \\
[\lambda y.s{\to}m]C\langle[m{\otimes}v, x]t\rangle & \mapsto_{\multimap} & C\langle[v{\to}y]L\langle[v'{\to}x]t\rangle\rangle & \text{with } s = L\langle v'\rangle
\end{array}
$$

Micro-step root exponential rules

$$
\begin{array}{rcll}
[v_{\mathtt{e}}{\to}e]C\langle\!\langle e\rangle\!\rangle & \mapsto_{\mathtt{axe}_1} & [v_{\mathtt{e}}{\to}e]C\langle\!\langle v_{\mathtt{e}}\rangle\!\rangle & \\
[f{\to}e]C\langle[e?x]t\rangle & \mapsto_{\mathtt{axe}_2} & [f{\to}e]C\langle[f?x]t\rangle & \\
[!s{\to}e]C\langle[e?x]t\rangle & \mapsto_{!} & [!s{\to}e]C\langle L\langle[v{\to}x]t\rangle\rangle & \text{with } s = L\langle v\rangle \\
[v_{\mathtt{e}}{\to}e]t & \mapsto_{\mathtt{w}} & t & \text{if } e \notin \mathtt{fv}(t)
\end{array}
$$

Contextual closure $\dfrac{t \mapsto_a s}{C\langle t\rangle \to_a C\langle s\rangle}$ for $a \in \{\mathtt{axm}_1, \mathtt{axm}_2, \multimap, \mathtt{axe}_1, \mathtt{axe}_2, !, \mathtt{w}\}$

Notations

$$
\begin{array}{rcl}
\text{Multiplicative} \quad \to_{\mathtt{m}} & := & \to_{\mathtt{axm}_1} \cup \to_{\mathtt{axm}_2} \cup \to_{\multimap} \\
\text{Exponential} \quad \to_{\mathtt{e}} & := & \to_{\mathtt{axe}_1} \cup \to_{\mathtt{axe}_2} \cup \to_{!} \cup \to_{\mathtt{w}} \\
\text{ESC} \quad \to_{\mathtt{ESC}} & := & \to_{\mathtt{m}} \cup \to_{\mathtt{e}} \\
\text{Non-erasing ESC} \quad \to_{\mathtt{ESC}_{-\mathtt{w}}} & := & \to_{\mathtt{m}} \cup \to_{\mathtt{axe}_1} \cup \to_{\mathtt{axe}_2} \cup \to_{!}
\end{array}
$$

Figure 1 The Exponential Substitution Calculus (ESC).

A fact used pervasively is that every term t writes, or *splits* uniquely as $t = L\langle v\rangle$. For instance $[e?m][v{\to}n]\lambda g.[m{\otimes}n, f]f$ splits as $L = [e?m][v{\to}n]\langle\cdot\rangle$ and $v = \lambda g.[m{\otimes}n, f]f$.

Because of split cuts and subtractions, the definition of plugging $C\langle t\rangle$ (or $C\langle D\rangle$) of a term t (or a context D) in a context C is slightly tricky, as it has to preserve the split shape. We refer the reader to [4] for such details, the definition is mostly as expected (the only two subtle cases are for $[\langle\cdot\rangle{\to}x]t$ and $[m{\otimes}\langle\cdot\rangle, x]$). Plugging can capture variables and we use $C\langle\!\langle t\rangle\!\rangle$ when we want to prevent it.

Types. The formulas of IMELL, and the deductive rules of the sequent calculus annotated with ESC terms, are in Fig. 2. They are taken directly from [4]. The typing system for ESC is exactly the standard sequent calculus for IMELL. Both formulas and rules are standard, but for the decoration with proof terms and the side conditions about variable names of the form $\Gamma\#\Delta$, which is a shortcut for $\mathtt{dom}\Gamma \cap \mathtt{dom}\Delta = \emptyset$. Linear implication $\multimap$ is also referred to as *lolli*. The only atomic formula that we consider, $X_{\mathtt{m}}$, is multiplicative. There is no multiplicative unit because, in presence of the exponentials, 1 can be simulated by $!(X_{\mathtt{m}} \multimap X_{\mathtt{m}})$. We distinguish between multiplicative and exponential axioms, in order to decorate them with the corresponding kind of variable.

Note that the weakening and contraction rules do not add constructors to terms. This is crucial in order to keep the calculus manageable. Note also that the decorations of the cut and $\multimap_l$ rules are *split*, as explained in the previous section. Clearly, typed terms are proper.

In this paper, types are only referred to in Lemma 3 below.

Multiplicative Cut Elimination Rules. The rewriting rules are in Fig. 1. The ESC has three multiplicative rules, in particular two for axioms, depending on whether they are acted upon ($\to_{\mathtt{axm}_1}$) or used to rename another multiplicative (thus linear) variable ($\to_{\mathtt{axm}_2}$). Rule

$$\text{Formulas} \quad A, B, C \quad ::= \quad X_{\mathtt{m}} \mid A \otimes B \mid A \multimap B \mid !A$$

Multiplicative rules

$$\frac{A \neq !B}{m : A \vdash m : A}\ \mathtt{ax_m} \qquad \frac{x : A, \Gamma \vdash t : B}{\Gamma \vdash \lambda x.t : A \multimap B}\ \multimap_r$$

$$\frac{\Gamma \vdash L\langle v\rangle : A \qquad \Delta, x : B \vdash t : C \qquad \Gamma \# (\Delta, x : B),\ m \text{ fresh}}{\Gamma, \Delta, m : A \multimap B \vdash L\langle [m \otimes v, x]t\rangle : C}\ \multimap_l$$

Exponential rules

$$\frac{}{e : !A \vdash e : !A}\ \mathtt{ax_e}$$

$$\frac{\Gamma, x : A \vdash t : B \qquad e \text{ fresh}}{\Gamma, e : !A \vdash [e?x]t : B}\ !_l \qquad \frac{!\Gamma \vdash t : A}{!\Gamma \vdash !t : !A}\ !_r$$

$$\frac{\Gamma \vdash t : A \qquad e \text{ fresh}}{\Gamma, e : !B \vdash t : A}\ \mathtt{w} \qquad \frac{\Gamma, e : !B, f : !B \vdash t : A}{\Gamma, e : !B \vdash \{e \to f\}t : A}\ \mathtt{c}$$

Cut

$$\frac{\Gamma \vdash L\langle v\rangle : A \qquad \Delta, x : A \vdash t : B \qquad \Gamma \# (\Delta, x : A)}{\Gamma, \Delta \vdash L\langle [v \to x]t\rangle : B}\ \mathtt{cut}$$

Figure 2 IMELL has a type system for ESC.

$\to_{\mathtt{axm}_1}$ is expressed generically for multiplicative values $v_{\mathtt{m}}$ (that is, multiplicative variables m and abstractions $\lambda x.t$). In $\to_{\mathtt{axm}_1}$, $\to_{\mathtt{axm}_2}$, and $\to_{\multimap}$, it is silently assumed that C does not capture m in $\to_{\mathtt{axm}_2}$ and $\to_{\multimap}$ (what is noted $C\langle\!\langle m\rangle\!\rangle$ in $\to_{\mathtt{axm}_1}$, while $\langle\!\langle\cdot\rangle\!\rangle$ is not used in $\to_{\mathtt{axm}_2}$, and $\to_{\multimap}$ because C might capture other variables in t and v). Note that, since C cannot capture m in these rules and terms are assumed to be proper, the hole of C cannot be contained in a !; this kind of context is called a *multiplicative context* in [4].

In $\to_{\multimap}$, the rule has to respect split cuts, which is why, for writing the reduct, the sub-term s is split on-the-fly. An example of $\to_{\multimap}$ step follows:

$$[\lambda e.[e?m]m \to n][n \otimes !f, o]o \quad \to_{\multimap} \quad [!f \to e][e?m][m \to o]o.$$

Exponential Rules. There are also four exponential rules, with again two rules for axioms. Replacement of variables ($\to_{\mathtt{axe}_1}$) and erasure ($\to_{\mathtt{w}}$) are expressed generically for exponential values $v_{\mathtt{e}}$ (that is, exponential variables f and promotions $!t$), interaction with derelictions (in $\to_{\mathtt{axe}_2}$ and $\to_!$) instead requires inspecting $v_{\mathtt{e}}$. Rule $\to_!$ removes the dereliction, copies the promotion body, and puts it in a cut – in proof nets jargon, *it opens the box*. To preserve the split shape, the body of the promotion is split and only the value is cut. An example:

$$[![f \to g]g \to e]\lambda m.[e?e'][m \otimes e', n]n \quad \to_! \quad [![f \to g]g \to e]\lambda m.[f \to g][g \to e'][m \otimes e', n]n.$$

Note that $\to_!$ entangles *interaction with a dereliction* and *duplication*, which is not what proof nets usually do (but this is what the LSC does). It is silently assumed that C does not capture e in $\to_{\mathtt{axe}_2}$ and $\to_!$ but it might capture other variables in t.

Comments about the Rewriting Rules. Unsurprisingly, proper terms are stable by reduction [4]. We also use a notion of position in terms.

▶ **Definition 1** (Positions). *A* position *in a term t is a decomposition $t = C\langle s\rangle$ such that s is a sub-term of t.*

For later defining the good strategy, we identify a redex with its position, which is a context. Every step $t \to_{\mathtt{ESC}_{\neg \mathtt{w}}} s$ reduces a redex of shape $t = C\langle [v{\to}x]D\langle t_x\rangle\rangle$ where t_x is an *occurrence* of x, *i.e.* a sub-term of t of shape x, $[x{\otimes}v, y]s$, or $[x?y]s$. The *redex position of $\to_{\mathtt{ESC}_{\neg \mathtt{w}}}$ steps* is the context $C\langle [v_{\mathtt{e}}{\to}x]D\rangle$. The *redex position of $\to_{\mathtt{w}}$ steps* is the context closing the root step. We write $C : t \to_{\mathtt{ESC}} s$ for a redex of position C in t.

Clashes. The presence of many constructors in an untyped setting gives rise to *clashes*, that is, irreducible cuts.

▶ **Definition 2** (Clashing, smooth, clash-free terms). *A* clash *is a term of the form $[v_{\mathtt{m}}{\to}e]t$ or $[v_{\mathtt{e}}{\to}m]t$. A term t is* clashing *if it has a clash as sub-term. Moreover, t is* clash-free *if, co-inductively, t is not clashing and s is clash-free for any step $t \to_{\mathtt{ESC}} s$.*

Clash-freeness is guaranteed by all forms of types (simple, polymorphic, recursive, and multi types, for instance), in particular by the IMELL types in Fig. 2, as ensured by the next lemma. Here, we adopt the untyped calculus but consider only clash-free terms, as in [4].

▶ **Lemma 3** ([4]). *Let t be a proper term.*
1. *If t has no clashes and $t \not\to_{\mathtt{ESC}}$ then t is cut-free.*
2. *If t is typable (by the IMELL type system in Fig. 2) then t is clash-free.*

Postponement of Garbage Collection. The erasing rule $\to_{\mathtt{w}}$, that models garbage collection, can be postponed. This point shall be crucial for our study.

▶ **Proposition 4** (Postponement of garbage collection, [4]). *If $t \to^*_{\mathtt{ESC}} s$ then $t \to^*_{\mathtt{ESC}_{\neg \mathtt{w}}} \to^*_{\mathtt{w}} s$.*

Basic Evaluation. As an intermediary step towards our results, we consider also a restricted form of ESC evaluation, forbidding reduction under all constructors but the right sub-terms of cuts, and dub it *basic evaluation*. It is one possible ESC analogous of weak evaluation in the λ-calculus (which forbids evaluation under abstraction). In λ-calculi with explicit substitutions, weak normal forms of closed terms are *answers*, i.e. abstractions possibly surrounded by explicit substitutions. We obtain a similar property for basic evaluation.

▶ **Definition 5** (Basic evaluation, answer). *A step $C : t \to_{\mathtt{ESC}} s$ (that is, a redex of position C in t) is* basic *if C is a cut context, which is also noted with $t \to_{\mathtt{b}} s$, or $t \to_{\mathtt{b}a} s$ with $a \in \{\mathtt{axm}_1, \mathtt{axm}_2, \multimap, \mathtt{axe}_1, \mathtt{axe}_2, !, \mathtt{w}\}$ if we want to specify the kind of step. Moreover, $\to_{\mathtt{b}\neg\mathtt{w}}$ denotes a non-erasing (that is, not $\mathtt{w}$) basic step. A term t is an* answer *if $t = E\langle v\rangle$ with E a cut context and v not a variable.*

▶ **Lemma 6.** *Let t be a closed clash-free term. Either t has a $\to_{\mathtt{b}\neg\mathtt{w}}$-step or it is an answer.*

Basic evaluation is enough to simulate in ESC the call-by-name/value weak evaluation of closed λ-terms, via Girard's translations to linear logic. Moreover, basic non-erasing evaluation is deterministic (and diamond, a property defined in Sect. 3, when erasing steps are considered). We omit the proofs of these facts, because neither basic evaluation nor λ-terms are the focus of the paper.

Dominating free variables of contexts

$$\begin{array}{rcl|rcl}
\mathtt{dv}(\langle\cdot\rangle) & := & \emptyset & \mathtt{dv}([m\otimes V, x]t) & := & \{m\}\cup\mathtt{dv}(V)\\
\mathtt{dv}(\lambda x.C) & := & \mathtt{dv}(C)\backslash\{x\} & \mathtt{dv}([m\otimes v, x]C) & := & \begin{cases}\{m\}\cup(\mathtt{dv}(C)\backslash\{x\}) & \text{if } x\in\mathtt{dv}(C)\\ \mathtt{dv}(C) & \text{otherwise.}\end{cases}\\
\mathtt{dv}(!C) & := & \mathtt{dv}(C) & & & \\
\mathtt{dv}([v\to x]C) & := & \mathtt{dv}(C)\backslash\{x\} & \mathtt{dv}([e?x]C) & := & \begin{cases}\{e\}\cup(\mathtt{dv}(C)\backslash\{x\}) & \text{if } x\in\mathtt{dv}(C)\\ \mathtt{dv}(C) & \text{otherwise.}\end{cases}\\
\mathtt{dv}([V\to x]t) & := & \mathtt{dv}(V) & & &
\end{array}$$

Good value contexts

$$V_G ::= \langle\cdot\rangle \mid \lambda x.G \mid\ !G$$

Good contexts

$$G ::= V_G \mid [m\otimes v, x]G \mid [m\otimes V_G, x]t \mid [e?x]G \mid [v\to x]G \text{ if } x\notin\mathtt{dv}(G)$$

Figure 3 Definitions for the good strategy: dominating free variables and good contexts.

Out Cuts and Garbage Collection. We shall mainly deal with cuts that are not contained in any other cut, the *out cuts*, which induce a notion of *out variable*.

▶ **Definition 7** (Out cuts, out variables). *The* out cuts *of a term t are those cuts in t that are not contained in any other cut, that is, if $t = C\langle[v\to x]s\rangle$ then $[v\to x]s$ is an out cut of t if C cannot be written as $C = D\langle[V\to y]u\rangle$ for some D, V, y, and u. The set of* out variables $\mathtt{ov}(t)$ *of t contains the variables having at least one occurrence out of all cuts of t.*

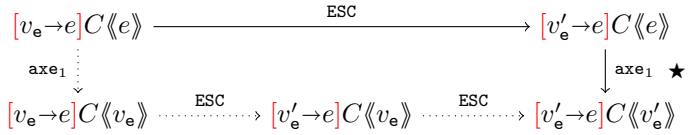
Out variables allow us to define a lax notion of cut-free terms where there are some cuts, but they can only be garbage collection cuts, or concern variable occurrences contained in garbage, so that the term becomes cut-free after garbage collection.

▶ **Definition 8** (Cut-freeness up to garbage). *A term t is cut-free up to garbage if $x\notin\mathtt{ov}(s)$ for all the out cuts $[v\to x]s$ of t.*

3 The Good Strategy

Here, we define the *good cut elimination strategy* $\to_{\mathtt{G}}$ for ESC, recalling the discussion from [4] that explains the design of the strategy. The strategy is conceived as to have the *sub-term property*, which is crucial for time analyses and which is defined as follows: every duplicated (or erased) sub-term during a cut elimination sequence is a sub-term of the initial term (up to variable renamings). See [4] for extensive discussions about the sub-term property.

Breaking the Sub-Term Property. When does the sub-term property not hold? One has to duplicate an exponential value $v_{\mathtt{e}}$ *touched* by previous steps. In our setting, *touched* can mean two things. Either a redex *fully* contained in $v_{\mathtt{e}}$ is reduced, obtaining $v'_{\mathtt{e}}$, and then $v'_{\mathtt{e}}$ is duplicated (or erased), as in the step marked with ★ in the following diagram (the other, dashed path of which has the sub-term property):



Preventing these situations from happening, thus forcing evaluation to follow the other (dashed) side of the diagram, is easy. It is enough to forbid the position of the reduced redex

to be inside the left sub-term of a cut – we say inside a *cut value* for short. It is however not enough, because cuts are also *created*. Consider:

$$[\lambda e.e{\to}m][m{\otimes}v_{\mathsf{e}},x]t \;\to_{\mathtt{ESC}}\; [\lambda e.e{\to}m][m{\otimes}v'_{\mathsf{e}},x]t \;\to_{\multimap}\; [v'_{\mathsf{e}}{\to}e][e{\to}x]t \;\overset{\star}{\to}_{\mathtt{axe}_1}\; [v'_{\mathsf{e}}{\to}e][v'_{\mathsf{e}}{\to}x]t$$

Reducing inside the subtraction value v_{e} leads to a *later* breaking of the sub-term property by the $\mathtt{axe}_1$ step, because the $\to_{\multimap}$ step creates a cut with v'_{e} inside. Preventing these cases is tricky, because forbidding reducing subtraction values leads to cut elimination stopping too soon, without producing a cut-free term. In the λ-calculus, it corresponds to forbidding reducing inside arguments, which leads to *head* reduction, that does not compute normal λ-terms. We shall then forbid reducing only subtraction values which are *at risk* of becoming cuts. In λ-calculus, leftmost reduction does reduce arguments but only when the left sub-term of the application is normal and not an abstraction, so that the argument is not involved in a β-redex. We shall do something similar here, but the sequent calculus formalizes this constraint differently, by checking that some variables are not captured.

Dominating Variables. The key notion is the one of *dominating (free) variables* $\mathtt{dv}(C)$ of a context (where C is meant to be the position of a redex), defined in Fig. 3, the base case of which is for $[m{\otimes}V,x]t$. If C is a position and $x \in \mathtt{dv}(C)$ then $[v{\to}x]C$ turns C into a dangerous position, that is, a redex of position $[v{\to}x]C : t \to_{\mathtt{ESC}} s$ might lead to a breaking of the sub-term property later on during cut elimination. In the example, e belongs to $\mathtt{dv}(C)$ for every context $C := [e?m][m{\otimes}V,x]s$ of $[e?m][m{\otimes}v,x]s$, for every V.

The Good Strategy. These considerations lead to the notion of *good contexts* in Fig. 3. A good context forbids the two ways of breaking the sub-term property: its hole cannot be in a cut value (note the absence of the production $[V_G{\to}x]t$) nor in a subtraction value such that one of its dominating variables is cut (because of the production $[v{\to}x]G$ if $x \notin \mathtt{dv}(G)$).

▶ **Definition 9** (Good steps, good strategy). *A step $C : t \to_{\mathtt{ESC}} s$ is* good *if its position C is good. In such a case, we write $t \to_{\mathtt{G}} s$. The* good cut elimination strategy *is simply $\to_{\mathtt{G}}$. We also use $\to_{\mathtt{G}a}$ to stress that the good step is of kind $a \in \{\mathtt{axm}_1, \mathtt{axm}_2, \multimap, \mathtt{axe}_1, \mathtt{axe}_2, !, \mathtt{w}\}$. The non-erasing good strategy $\to_{\mathtt{G}_{\neg\mathtt{w}}}$ is the variant of $\to_{\mathtt{G}}$ excluding $\to_{\mathtt{Gw}}$ steps.*

The sub-term property in the following theorem captures the quantitative aspect for cost analyses, i.e. the bound on the size of duplicated values by the size of the initial term.

▶ **Theorem 10** (Properties of the good strategy, [4]).

1. Quantitative sub-term property*: if $e : t \to_{\mathtt{G}}^{*} s$ and v be a value erased or duplicated along e, then $|v| \leq |t|$.*
2. Diamond*: if $s_1 \;{}_{\mathtt{G}}{\leftarrow} t \to_{\mathtt{G}} s_2$ and $s_1 \neq s_2$ then $s_1 \to_{\mathtt{G}} u \;{}_{\mathtt{G}}{\leftarrow} s_2$ for some u.*
3. Fullness*: if t is clash-free and not $\to_{\mathtt{ESC}}$-normal then $t \to_{\mathtt{G}} s$ for some s.*
4. Good polynomial cost model*: if $\Gamma \vdash t : A$ is a typed term then there exist k and a cut-free term s such that $t \to_{\mathtt{G}}^{k} s$, and such a reduction sequence is implementable on RAMs in time polynomial in k and $|t|$.*

The algorithmic aspect of Theorem 10.4 is proved in [4] via arguments that do not establish the degree of the polynomial bound. The aim of this paper is exactly to show that a carefully designed abstract machine provides a bound that is linear in both k and $|t|$.

The Diamond Property. Let us provide some background about the diamond property. Following Dal Lago and Martini [34], we say that a relation $\to_{\mathtt{r}}$ is *diamond* if $s_1 \ {}_{\mathtt{r}}\!\leftarrow t \to_{\mathtt{r}} s_2$ and $s_1 \neq s_2$ imply $s_1 \to_{\mathtt{r}} u \ {}_{\mathtt{r}}\!\leftarrow s_2$ for some u. The terminology in the literature is inconsistent: Terese [44, Exercise 1.3.18] dubs this property CR[1], and defines the diamond more restrictively, without requiring $s_1 \neq s_2$ in the hypothesis: s_1 and s_2 have to join even if $s_1 = s_2$.

Standard corollaries of Dal Lago and Martini's notion are that, if $\to_{\mathtt{r}}$ is diamond, then:

1. *Confluence*: $\to_{\mathtt{r}}$ is confluent, that is, $s_1 \ {}^{*}_{\mathtt{r}}\!\leftarrow t \to^{*}_{\mathtt{r}} s_2$ implies $s_1 \to^{*}_{\mathtt{r}} u \ {}^{*}_{\mathtt{r}}\!\leftarrow s_2$ for some u;
2. *Length invariance*: all $\mathtt{r}$-evaluations with the same start and $\mathtt{r}$-normal end terms have the same length (*i.e.* if $e\colon t \to^{k}_{\mathtt{r}} s$ and $e'\colon t \to^{h}_{\mathtt{r}} s$ with s $\mathtt{r}$-normal, then $h = k$);
3. *Uniform normalization*: t is weakly $\mathtt{r}$-normalizing if and only if it is strongly $\mathtt{r}$-normalizing.

Basically, the diamond property captures a more liberal form of determinism. In particular, length invariance is essential in order to take the number of steps of a strategy as a cost model. Without it, indeed, the number of steps of a non-deterministic strategy would be an ambiguously defined notion of cost.

4 Preliminaries on Abstract Machines

Abstract Machines Glossary. Abstract machines manipulate *pre-terms*, that is, terms without implicit α-renaming. In this paper, an *abstract machine* is a quadruple $\mathtt{M} = (\mathtt{States}, \rightsquigarrow, \cdot \triangleleft \cdot, \underline{\cdot})$ the components of which are as follows.

- *States.* A state $Q \in \mathtt{States}$ is composed by the *active term* t, plus one data structure which depends on the actual machine. Terms in states are actually pre-terms.
- *Transitions.* The pair $(\mathtt{States}, \rightsquigarrow)$ is a transition system with transitions $\rightsquigarrow$ partitioned into *principal transitions*, whose union is noted $\rightsquigarrow_{\mathtt{pr}}$ and that are meant to correspond to cut-elimination steps on the calculus, and *search transitions*, whose union is noted $\rightsquigarrow_{sea}$, that take care of searching for (principal) redexes.
- *Initialization.* The component $\triangleleft \subseteq \Lambda \times \mathtt{States}$ is the *initialization relation* associating terms to initial states. It is a *relation* and not a function because $t \triangleleft Q$ maps a λ-term t (considered modulo α) to a state Q having a *pre-term representant* of t (which is not modulo α) as active term. Intuitively, any two states Q and Q' such that $t \triangleleft Q$ and $t \triangleleft Q'$ are α-equivalent. The initializing terms (i.e. those t such that $t \triangleleft Q$ for some Q) are always *proper* and *clash-free*. A state Q is *reachable* if it can be reached starting from an initial state, that is, if $Q' \rightsquigarrow^{*} Q$ where $t \triangleleft Q'$ for some t and Q', shortened as $t \triangleleft Q' \rightsquigarrow^{*} Q$.
- *Read-back.* The read-back function $\underline{\cdot} : \mathtt{States} \to \Lambda$ turns reachable states into terms and satisfies the *initialization constraint*: if $t \triangleleft Q$ then $\underline{Q} =_{\alpha} t$.

Further Terminology and Notations. A state is *final* if no transitions apply. A *run* $r : Q \rightsquigarrow^{*} Q'$ is a possibly empty finite sequence of transitions, the length of which is noted $|r|$; note that the first and the last states of a run are not necessarily initial and final. If a and b are transitions labels (that is, $\rightsquigarrow_a \subseteq \rightsquigarrow$ and $\rightsquigarrow_b \subseteq \rightsquigarrow$) then $\rightsquigarrow_{a,b} := \rightsquigarrow_a \cup \rightsquigarrow_b$ and $|r|_a$ is the number of a transitions in r, and $|r|_{\neg a}$ is the number of transitions in r that are not $\rightsquigarrow_a$.

Well-Boundness and Renamings. For the machines at work in this paper, the pre-terms in initial states shall be *well-bound*, that is, they have pairwise distinct bound names; for instance $[m \otimes \lambda n.n, e]\lambda f.f$ is well-bound while $[m \otimes \lambda n.n, e]\lambda n.n$ is not. We shall also write t^{α} in a state Q for a *fresh well-bound renaming* of t, *i.e.* t^{α} is α-equivalent to t, well-bound, and its bound variables are fresh with respect to those in t and in the other components of Q.

Implementation Theorem, Abstractly. We now formally define the notion of a machine implementing a strategy. Since the good strategy is non-deterministically diamond but the machine that shall implement it is deterministic, we need a slightly unusual form of implementation theorem. As it is standard, machine transitions shall be mapped to equalities or steps on the calculus. For the other direction, however, we obtain only a *big-step* simulation, that is, if the strategy on the calculus terminates / diverges then the same does the machine, and with a related number of steps. But there is no step-by-step simulation of the calculus by the machine, because the non-deterministic strategy might do a step which is not the one done by the machine. Everything works fine because of the properties of diamond strategies. The general scheme is inspired by Accattoli et al.'s scheme for strong call-by-value [12], but it is here simpler because of the absence of structural equivalence and implosive sharing.

It would also be possible to have a more symmetric setting by either designing a more complex diamond machine, as done by Accattoli and Barenbaum [5], or by designing a deterministic variant of the good strategy. For the sake of simplicity, we prefer the asymmetric setting as to keep a simple machine and reuse the notion of good strategy from the literature.

▶ **Definition 11** (Big-step implementations). *A machine* $\mathtt{M} = (\mathtt{States}, \rightsquigarrow, \cdot \triangleleft \cdot, \underline{\cdot})$ *is a* big-step implementation of a strategy $\to_{\mathtt{str}}$ *on terms when, given a (proper and clash-free) term* t*:*

1. Runs to evaluations*: for any* $\mathtt{M}$*-run* $r : t \triangleleft Q' \rightsquigarrow^* Q$ *there is a* $\to_{\mathtt{str}}$*-evaluation* $e : t \to^*_{\mathtt{str}} \underline{Q}$ *with* $|r|_{\mathtt{pr}} = |e|$.
2. Normalizing evaluations to runs*: if* $e \colon t \to^*_{\mathtt{str}} s$ *with* s $\to_{\mathtt{str}}$*-normal then there is a* $\mathtt{M}$*-run* $r \colon t \triangleleft Q' \rightsquigarrow^* Q$ *such that* $\underline{Q} = s$ *with* $|r|_{\mathtt{pr}} = |e|$.
3. Diverging evaluations to runs*: if* $t \triangleleft Q$ *and* $\to_{\mathtt{str}}$ *diverges on* t *then* $\mathtt{M}$ *diverges on* Q *doing infinitely many principal transitions.*

Next, we isolate sufficient conditions for big-step implementations.

▶ **Definition 12** (Lax implementation system). *A lax implementation system is given by a machine* $\mathtt{M} = (\mathtt{States}, \rightsquigarrow, \cdot \triangleleft \cdot, \underline{\cdot})$ *and a strategy* $\to_{\mathtt{str}}$ *such that for every reachable state* Q*:*

1. Principal projection*: if* $Q \rightsquigarrow_{\mathtt{pr}} Q'$ *then* $\underline{Q} \to_{\mathtt{str}} \underline{Q'}$*;*
2. Search transparency*: if* $Q \rightsquigarrow_{sea} Q'$ *then* $\underline{Q} = \underline{Q'}$*;*
3. Search terminates*:* $\rightsquigarrow_{sea}$ *terminates;*
4. Halt*: if* Q *is final then* $\underline{Q}$ *is* $\to_{\mathtt{str}}$*-normal;*
5. Diamond*:* $\to_{\mathtt{str}}$ *is diamond.*

▶ **Theorem 13** (Abstract big-step implementation). *Let* $\mathtt{M}$ *and* $\to_{\mathtt{str}}$ *form a lax implementation system. Then,* $\mathtt{M}$ *is a big-step implementation of* $\to_{\mathtt{str}}$*.*

Clash-Free and Proper States. Note that we have *not* taken into account properness and clashes on states. One can say that a state Q is proper (resp. clash-free) if its decoding $\underline{Q}$ is. In this way, these notions are trivially seen to be preserved by runs in a lax implementation system. Note, indeed, that transitions are mapped via decoding to either equalities or rewriting steps (and the proofs of these facts shall not need properness nor clash-freeness). Since states are initialized with proper and clash-free terms, and that these notions on terms are preserved by reduction (clash-freeness by definition, for properness see [4]), proper and clash-free states are preserved by transition. Therefore, we shall omit all considerations about properness and clashes for machines.

$$\begin{array}{rrcl} \textsc{States} & Q, Q' & ::= & (E \mid t) \\ \textsc{Read back} & \underline{(E \mid t)} & := & E\langle t\rangle \end{array} \qquad \textsc{Initialization} \quad t \triangleleft Q \quad \text{if} \quad Q = (\langle\cdot\rangle \mid t^{\alpha})$$

Cut Ctx	Active Tm	Tran.	Cut Ctx	Active Tm	
E	$[v{\to}x]t$	$\rightsquigarrow_{sea}$	$E\langle[v{\to}x]\langle\cdot\rangle\rangle$	t	
$E\langle[n{\to}m]E'\rangle$	$[m{\otimes}v, x]t$	$\rightsquigarrow_{\mathtt{axm}_2}$	$E\langle E'\rangle$	$[n{\otimes}v, x]t$	
$E\langle[\lambda y.L\langle v'\rangle{\to}m]E'\rangle$	$[m{\otimes}v, x]t$	$\rightsquigarrow_{\multimap}$	$E\langle E'[v{\to}y]\rangle$	$L\langle v'{\to}x]t\rangle$	
$E\langle[f{\to}e]E'\rangle$	$[e?x]t$	$\rightsquigarrow_{\mathtt{axe}_2}$	$E\langle[f{\to}e]E'\rangle$	$[f?x]t$	
$E\langle[!L\langle v\rangle{\to}e]E'\rangle$	$[e?x]t$	$\rightsquigarrow_{!}$	$E\langle[!L\langle v\rangle{\to}e]E'\rangle$	$L'\langle[v'{\to}x]t\rangle$	#
$E\langle[v_{\mathtt{m}}{\to}m]E'\rangle$	m	$\rightsquigarrow_{\mathtt{axm}_1}$	$E\langle E'\rangle$	$v_{\mathtt{m}}$	
$E\langle[v_{\mathtt{e}}{\to}e]E'\rangle$	e	$\rightsquigarrow_{\mathtt{axe}_1}$	$E\langle[v_{\mathtt{e}}{\to}e]E'\rangle$	$v_{\mathtt{e}}^{\alpha}$	

with $L'\langle v'\rangle = L\langle v\rangle^{\alpha}$

Figure 4 The Basic Abstract Machine (BAM).

5 A Machine for the Closed Basic Case

Here, we study a machine implementing basic evaluation on closed ESC terms. The aim is to give a gentle introduction to some machine concepts.

BAM. The *basic abstract machine* (BAM) is defined in Fig. 4. States $Q = (E \mid t)$ are simply given by the active (pre-)term t and a cut context E which is a list containing the cuts encountered so far by the search mechanism, playing the role of the (global) environment in machines such as Accattoli et al.'s Milner Abstract Machine [6].

The initialization relation $t \triangleleft (\langle\cdot\rangle \mid t^{\alpha})$ pairs (proper and clash-free) terms t with states composed by an empty cut context $\langle\cdot\rangle$ and a well-bound renaming t^{α} of t. The BAM has six principal transitions, mimicking the ESC rewriting rules but for the weakening one (BAM never erases), plus one search transition $\rightsquigarrow_{sea}$, moving cuts from the term to the cut context.

The BAM looks at the topmost constructor of the active term and proceeds applying a transition belonging to one of the following three groups. If the constructor is:

- *Search*: a cut $[v{\to}x]$, the BAM adds it to the cut context, by applying $\rightsquigarrow_{sea}$;
- *Computation*: a subtraction $[m{\otimes}v, x]$ or a dereliction $[e?x]$, then the BAM looks for the associated cut in the cut context, it applies the corresponding cut elimination rule, and goes on to execute the modified active term, by applying $\rightsquigarrow_{\mathtt{axm}_2}$, $\rightsquigarrow_{\multimap}$, $\rightsquigarrow_{\mathtt{axe}_2}$, or $\rightsquigarrow_{!}$;
- *Terminal replacements*: a variable m or e, the BAM looks for the associated cut in the cut context and it applies the matching replacement via $\rightsquigarrow_{\mathtt{axm}_1}$ or $\rightsquigarrow_{\mathtt{axe}_1}$. Note that terminal replacements can only be followed by further terminal replacements, hence the name.

Note that in transitions $\rightsquigarrow_{!}$ and $\rightsquigarrow_{\mathtt{axe}_1}$ some renaming takes place, using names that are fresh with respect to the whole state. The domain $\mathtt{dom}(E)$ of a cut context is defined as $\mathtt{dom}(\langle\cdot\rangle) := \emptyset$ and $\mathtt{dom}([v{\to}x]E) := \{x\} \cup \mathtt{dom}(E)$.

▶ **Lemma 14** (BAM qualitative invariants). *Let $Q = (E \mid t)$ be a BAM reachable state.*

1. Closure*: $\mathtt{fv}(t) \subseteq \mathtt{dom}(E)$ and if $E = E'\langle[v{\to}x]E''\rangle$ then $\mathtt{fv}(v) \subseteq \mathtt{dom}(E')$.*
2. Well-bound*: if $\lambda x.s$, $[e?x]s$, $[m{\otimes}v, x]s$, or $[v{\to}x]s$ occur in Q and x has any other occurrence in Q then it is a free variable of s, and if $E = E'\langle[v{\to}x]E''\rangle$ and x has any other occurrence in Q then it is a free variable in E'' or t.*

▶ **Proposition 15.** *Let Q be a BAM reachable state and $a \in \{\mathtt{axm}_1, \mathtt{axm}_2, \mathtt{axe}_1, \mathtt{axe}_2, \multimap, !\}$.*

1. Search transparency*: if $Q \rightsquigarrow_{sea} Q'$ then $\underline{Q} = \underline{Q'}$.*
2. Principal projection*: if $Q \rightsquigarrow_{a} Q'$ then $\underline{Q} \to_{\mathtt{ba}} \underline{Q'}$.*
3. Search termination*: transition $\rightsquigarrow_{sea}$ terminates.*

$$\begin{array}{rrcl|rrcl} \textsc{Pools} & P,P' & ::= & \epsilon \mid \langle t\rangle_a{:}P & \textsc{Init.} & t \triangleleft Q & \text{if} & Q = (\langle\cdot\rangle_a \mid t^\alpha) \\ \textsc{States} & Q,Q' & ::= & (\mathbb{C} \mid P) & \textsc{Read back} & \underline{(\mathbb{C} \mid P)} & := & \mathbb{C}\langle t_1\rangle_{a_1}\dots\langle t_k\rangle_{a_k} \\ & & & & & & & \text{with } P = \langle t_1\rangle_{a_1}{:}\dots{:}\langle t_k\rangle_{a_k}{:}\epsilon \end{array}$$

MCtx	Pool	Tran.	MCtx	Pool	
$\mathbb{C}$	$\langle[v{\to}x]t\rangle_a{:}P$	$\rightsquigarrow_{sea_1}$	$\mathbb{C}\langle[v{\to}x]\langle\cdot\rangle_a\rangle_a$	$\langle t\rangle_a{:}P$	
$\mathbb{C}$ with $m \notin \mathtt{dom}(\mathbb{C})$	$\langle[m{\otimes}v, x]t\rangle_a{:}P$	$\rightsquigarrow_{sea_2}$	$\mathbb{C}\langle[m{\otimes}\langle\cdot\rangle_b, x]\langle\cdot\rangle_a\rangle_a$	$\langle v\rangle_b{:}\langle t\rangle_a{:}P$	*
$\mathbb{C}$ with $e \notin \mathtt{dom}(\mathbb{C})$	$\langle[e?x]t\rangle_a{:}P$	$\rightsquigarrow_{sea_3}$	$\mathbb{C}\langle[e?x]\langle\cdot\rangle_a\rangle_a$	$\langle t\rangle_a{:}P$	
$\mathbb{C}$	$\langle\lambda x.t\rangle_a{:}P$	$\rightsquigarrow_{sea_4}$	$\mathbb{C}\langle\lambda x.\langle\cdot\rangle_a\rangle_a$	$\langle t\rangle_a{:}P$	
$\mathbb{C}$	$\langle!t\rangle_a{:}P$	$\rightsquigarrow_{sea_5}$	$\mathbb{C}\langle!\langle\cdot\rangle_a\rangle_a$	$\langle t\rangle_a{:}P$	
$\mathbb{C}$ with $x \notin \mathtt{dom}(\mathbb{C})$	$\langle x\rangle_a{:}P$	$\rightsquigarrow_{sea_6}$	$\mathbb{C}\langle x\rangle_a$	P	
$\mathbb{C}\langle[n{\to}m]\mathbb{C}'\rangle_\circ$	$\langle[m{\otimes}v, x]t\rangle_a{:}P$	$\rightsquigarrow_{\mathtt{axm}_2}$	$\mathbb{C}\langle\mathbb{C}'\rangle_\circ$	$\langle[n{\otimes}v, x]t\rangle_a{:}P$	
$\mathbb{C}\langle[\lambda y.L\langle v'\rangle{\to}m]\mathbb{C}'\rangle_\circ$	$\langle[m{\otimes}v, x]t\rangle_a{:}P$	$\rightsquigarrow_{\multimap}$	$\mathbb{C}\langle\mathbb{C}'\rangle_\circ$	$\langle[v{\to}y]L\langle[v'{\to}x]t\rangle\rangle_a{:}P$	
$\mathbb{C}\langle[f{\to}e]\mathbb{C}'\rangle_\circ$	$\langle[e?x]t\rangle_a{:}P$	$\rightsquigarrow_{\mathtt{axe}_2}$	$\mathbb{C}\langle[f{\to}e]\mathbb{C}'\rangle_\circ$	$\langle[f?x]t\rangle_a{:}P$	
$\mathbb{C}\langle[!L\langle v\rangle{\to}e]\mathbb{C}'\rangle_\circ$	$\langle[e?x]t\rangle_a{:}P$	$\rightsquigarrow_!$	$\mathbb{C}\langle[!L\langle v\rangle{\to}e]\mathbb{C}'\rangle_\circ$	$\langle L'\langle[v'{\to}x]t\rangle\rangle_a{:}P$	#
$\mathbb{C}\langle[v_{\mathtt{m}}{\to}m]\mathbb{C}'\rangle_\circ$	$\langle m\rangle_a{:}P$	$\rightsquigarrow_{\mathtt{axm}_1}$	$\mathbb{C}\langle\mathbb{C}'\rangle_\circ$	$\langle v_{\mathtt{m}}\rangle_a{:}P$	
$\mathbb{C}\langle[v_{\mathtt{e}}{\to}e]\mathbb{C}'\rangle_\circ$	$\langle e\rangle_a{:}P$	$\rightsquigarrow_{\mathtt{axe}_1}$	$\mathbb{C}\langle[v_{\mathtt{e}}{\to}e]\mathbb{C}'\rangle_\circ$	$\langle v_{\mathtt{e}}^\alpha\rangle_a{:}P$	

* b is fresh. # with $L'\langle v'\rangle = L\langle v\rangle^\alpha$

Figure 5 The Strong Exponential Substitution Abstract Machine without Erasure (SESAME).

4. Halt*: if Q is final then $Q = (E \mid v)$ with v not a variable, and $\underline{Q}$ is normal for non-erasing basic evaluation.*

▶ **Theorem 16.** *The BAM implements ESC non-erasing basic evaluation on closed terms.*

6 SESAME

In this section, we extend the BAM as to perform the analogous of strong evaluation, that is, as to perform cut-elimination also inside values, and implement the good strategy. The obtained machine shall ignore garbage collection and – when it terminates – it returns a term that is cut-free *up to garbage* (Def. 8). Garbage collection shall be addressed in Sect. 8.

Pools and Jobs. The *Strong Exponential Substitution Abstract Machine without Erasure* (SESAME), defined in Fig. 5, relies on a technique for strong evaluation recently introduced by Accattoli and Barenbaum [5], which we are now going to explain.

When evaluation goes under binders, the closure invariant of the basic case (Lemma 14.1), for which free variables of the active term are associated to a cut in the cut context, is lost. Thus, when the active term is a subtraction $[m{\otimes}v, x]t$ with no associated cut for m, the subtraction is kept, and the machine has to evaluate v and t. Now, the evaluations of v and t cannot affect each other, they are independent, but one of the two sub-terms has to be evaluated first, say v. Usually, strong machines (such as Crégut's [26]) would run through v, and if such process does terminate, producing a value v', then they *backtrack* to the subtraction generating the fork by *moving sequentially* through v', and then start evaluate t.

Accattoli and Barenbaum's technique simply circumvents the sequential backtracking process by directly jumping back to the forking point. For that, the machine is equipped with a *pool* P of *jobs*, each one paired to a unique name a. In our example, there would be a job $\langle v\rangle_a$ for v and a job $\langle t\rangle_b$ for t. The idea is that when one job ends then the machine jumps to the next one, *without* moving through the structure of the finished job.

In [5], the pool is any data structure satisfying a certain interface. The idea is that different data structures implementing the interface realize different job scheduling policies, compactly accounting for different strong strategies within the same framework. We here omit this abstract aspect and fix the pool to be the simplest such structure, namely a LIFO list of named jobs. In the λ-calculus, LIFO list pools implement leftmost evaluation [5].

Another simplification with respect to [5] is that here jobs are simply named terms, while in [5] they are named pairs of a term and an applicative stack. The change induces a simpler notion of read back. This difference is not a design choice, it is simply induced by having a machine for sequent calculus terms (here) rather than natural deduction terms (in [5]).

A crucial point of SESAME is that job forking happens on subtractions only, that is, it does not happen on cuts $[v{\to}x]t$: the machine ignores v and goes straight to evaluate t. This happens in particular to prevent the breaking of the sub-term property.

We write $\mathtt{names}(P)$ for the set of names associated to the jobs in P, that is, if $P = \langle t_1\rangle_{a_1}{:}\ldots{:}\langle t_k\rangle_{a_k}$ then $\mathtt{names}(P) := \{a_1,\ldots,a_k\}$.

Multi-Contexts and Approximants. Another aspect of the technique in [5] is that the parts of the term that have been evaluated and that shall not be touched again by the machine – sometimes referred to as *stable parts* – are accumulated in the *approximant* $\mathbb{A}$ (of the normal form), which is a multi-context (that is, a context with possibly many holes, possibly none). The idea is that the name a of a job $\langle t\rangle_a$ in the pool is associated to a (unique) named hole $\langle\cdot\rangle_a$ in $\mathbb{A}$, and that, whenever a stable piece of term is produced by the job $\langle t\rangle_a$, that piece is moved to $\langle\cdot\rangle_a$ in $\mathbb{A}$, incrementally building the normal form.

We follow this pattern from [5], but our setting induces a few differences. Firstly, in [5] the machine has also a global environment, akin to the cut context of the BAM. Here, we include the environment/cut context *into* the approximant, having only one data structure. Thus, the approximant shall have cuts, but these cuts bind variables that have non-garbage occurrences only in the active term, i.e., only out of the approximant itself, where instead they have only garbage occurrences. At the end of a complete run, the approximant shall be normal for the non-erasing good strategy, i.e. it shall be cut-free up to garbage (Def. 8).

Secondly, in [5] the machine is defined using approximants, while here we define it using the weaker notion of multi-contexts, and then prove invariants guaranteeing that the multi-contexts of reachable states are approximants.

▶ **Definition 17** (Multi-contexts and approximants). *A* (named) multi-context $\mathbb{C}$ *is an ESC term in which there might be occurrences of holes* $\langle\cdot\rangle_a$ *indexed with names, as follows:*

$$\begin{array}{l|l} \textsc{Names}\quad a,b,c,a_1,b_2,\ldots & \textsc{Value multi-ctxs}\quad \mathbb{V} ::= \langle\cdot\rangle_a \mid x \mid \lambda x.\mathbb{C} \mid\, !\mathbb{C} \\ & \textsc{Multi-ctxs}\quad \mathbb{C} ::= \mathbb{V} \mid [\mathbb{V}{\to}x]\mathbb{C} \mid [m{\otimes}\mathbb{V},x]\mathbb{C} \mid [e?x]\mathbb{C} \end{array}$$

The plugging $\mathbb{C}\langle\mathbb{C}'\rangle_a$ *of* $\mathbb{C}'$ *on* a *in* $\mathbb{C}$*, is the capture-allowing substitution of* $\langle\cdot\rangle_a$ *by* $\mathbb{C}'$ *in* $\mathbb{C}$*. We write* $\mathbb{C}\langle\mathbb{C}'\rangle_\circ$ *for when* $\circ$ *is an irrelevant name occurring exactly once in* $\mathbb{C}$*; this notation is meant to be used for decomposing a multi-context in two, as in* $\mathbb{C} = \mathbb{C}'\langle\mathbb{C}''\rangle_\circ$*. We write* $\mathtt{names}(\mathbb{C})$ *for the set of names that occur in* $\mathbb{C}$*. Out variables and out cuts extend to multi-contexts as expected.*

The domain $\mathtt{dom}(\mathbb{C}) = \{x_1,\ldots,x_n\}$ *of a multi-context* $\mathbb{C}$ *contains the variables on which there is a cut in* $\mathbb{C}$ *(the formal definition is in Appendix E of the tech report [11]).*

An approximant $\mathbb{A}$ *is a multi-context such that:*

1. Unique names*: every name* $a \in \mathtt{names}(\mathbb{A})$ *has exactly one occurrence in* $\mathbb{A}$*;*
2. Out cuts are hereditary garbage and hole-free*: for every out cut* $[\mathbb{V}{\to}x]\mathbb{C}'$ *in* $\mathbb{A}$*, if* $x \in \mathtt{fv}(\mathbb{C}')$ *then* x *occurs only inside cut values, i.e.* $x \notin \mathtt{ov}(\mathbb{C}')$*, and* $\mathbb{V}$ *is a term.*

Note that a multi-context $\mathbb{C}$ without holes is simply a term, thus the defined notion of plugging subsumes the plugging $\mathbb{C}\langle t\rangle_a$ of terms in multi-contexts.

Transitions. The principal transition of SESAME are as for the BAM up to the generalization of the cut context E to a multi-context $\mathbb{C}$. Note the mute name $\circ$ for decomposing the multi-context in the transitions. Clearly, the name $\circ$ on the LHS and the RHS of each transition is the same. SESAME has a search transition for each constructor of the calculus. Note that transitions $\rightsquigarrow_{\mathtt{axm}_1}$ and $\rightsquigarrow_{\mathtt{axe}_1}$ are no longer *terminal*, because now they might be followed by a search transition, after which the run might jump to a different job.

Read Back. The read back $\underline{Q}$ of a state $Q = (\mathbb{C} \mid P)$ is defined in Fig. 5 and simply plugs each job $\langle t \rangle_a$ of the pool P in the hole of name a of the multi context $\mathbb{C}$. We also write $\mathbb{C}\langle P \rangle$ for $\underline{Q}$. Since the holes of $\mathbb{C}$ are all independent and jobs contains terms (with no holes), we have that $\mathbb{C}\langle P \rangle = \mathbb{C}\langle P' \rangle$ for any pool P' obtained by permuting the elements of P.

Example of SESAME run. As an example, we show a SESAME run on the following term:

$$t \quad := \quad [!\lambda m_1.m_1 \to e_1][e_1 ? m_2][e_1 ? m_3][m_2 \ominus m_3, m_4]m_4$$

that just applies an identity function to itself, by making two copies of $!\lambda m_1.m_1$ and applying one to the other via the subtraction. To fit it into the margins, we use the abbreviation $s := [m_2 \ominus m_3, m_4]m_4$.

Tr.	MCtx	Pool
	$\langle \cdot \rangle_a$	$\langle t \rangle_a : \epsilon$
$\rightsquigarrow_{sea_1}$	$[!\lambda m_1.m_1 \to e_1]\langle \cdot \rangle_a$	$\langle [e_1 ? m_2][e_1 ? m_3]s \rangle_a : \epsilon$
$\rightsquigarrow_{!}$	$[!\lambda m_1.m_1 \to e_1]\langle \cdot \rangle_a$	$\langle [\lambda m_5.m_5 \to m_2][e_1 ? m_3]s \rangle_a : \epsilon$
$\rightsquigarrow_{sea_1}$	$[!\lambda m_1.m_1 \to e_1][\lambda m_5.m_5 \to m_2]\langle \cdot \rangle_a$	$\langle [e_1 ? m_3]s \rangle_a : \epsilon$
$\rightsquigarrow_{!}$	$[!\lambda m_1.m_1 \to e_1][\lambda m_5.m_5 \to m_2]\langle \cdot \rangle_a$	$\langle [\lambda m_6.m_6 \to m_3]s \rangle_a : \epsilon$
$\rightsquigarrow_{sea_1}$	$[!\lambda m_1.m_1 \to e_1][\lambda m_5.m_5 \to m_2][\lambda m_6.m_6 \to m_3]\langle \cdot \rangle_a$	$\langle s \rangle_a : \epsilon$
$\rightsquigarrow_{\multimap}$	$[!\lambda m_1.m_1 \to e_1][\lambda m_6.m_6 \to m_3]\langle \cdot \rangle_a$	$\langle [m_3 \to m_5][m_5 \to m_4]m_4 \rangle_a : \epsilon$
$\rightsquigarrow_{sea_1}$	$[!\lambda m_1.m_1 \to e_1][\lambda m_6.m_6 \to m_3][m_3 \to m_5]\langle \cdot \rangle_a$	$\langle [m_5 \to m_4]m_4 \rangle_a : \epsilon$
$\rightsquigarrow_{sea_1}$	$[!\lambda m_1.m_1 \to e_1][\lambda m_6.m_6 \to m_3][m_3 \to m_5][m_5 \to m_4]\langle \cdot \rangle_a$	$\langle m_4 \rangle_a : \epsilon$
$\rightsquigarrow_{\mathtt{axm}_1}$	$[!\lambda m_1.m_1 \to e_1][\lambda m_6.m_6 \to m_3][m_3 \to m_5]\langle \cdot \rangle_a$	$\langle m_5 \rangle_a : \epsilon$
$\rightsquigarrow_{\mathtt{axm}_1}$	$[!\lambda m_1.m_1 \to e_1][\lambda m_6.m_6 \to m_3]\langle \cdot \rangle_a$	$\langle m_3 \rangle_a : \epsilon$
$\rightsquigarrow_{\mathtt{axm}_1}$	$[!\lambda m_1.m_1 \to e_1]\langle \cdot \rangle_a$	$\langle \lambda m_6.m_6 \rangle_a : \epsilon$
$\rightsquigarrow_{sea_4}$	$[!\lambda m_1.m_1 \to e_1]\lambda m_6.\langle \cdot \rangle_a$	$\langle m_6 \rangle_a : \epsilon$
$\rightsquigarrow_{sea_6}$	$[!\lambda m_1.m_1 \to e_1]\lambda m_6.m_6$	ϵ

Note that in the last state the cut $[!\lambda m_1.m_1 \to e_1]$ is garbage but it is not removed. This shall be taken care of by an extra garbage collection phase described in Sect. 8.

7 Analysis of SESAME

Invariants. As for the BAM, we prove some invariants. The first one states that the multi context of a reachable state is an approximant. A second invariant states that names in the multi context are exactly those in the pool, where they have exactly one occurrence each. Then, we need a well-bound invariant about binders, relying on a technical definition in Appendix E of the tech report [11], along the lines of the invariant of the BAM. Lastly, the contextual decoding invariant, the most important and sophisticated invariant, states that a reachable state less the first job reads back to a good context. For the invariant to hold, the statement has to be generalized in a technical way, analogously to similar invariants in [5].

▶ **Lemma 18** (SESAME qualitative invariants). *Let $Q = (\mathbb{C} \mid P)$ be a SESAME state reachable from a well-bound initial term t_0.*

1. Approximant*: $\mathbb{C}$ is an approximant.*
2. Names*: jobs in the pool P have pairwise distinct names, and $\mathtt{names}(\mathbb{C}) = \mathtt{names}(P)$.*
3. Well-bound*: Q is a well-bound state.*
4. Contextual decoding*: if $P = \langle t_1\rangle_{a_1} : \ldots : \langle t_k\rangle_{a_k}$ has length $k \geq 1$ then:*

$$\mathbb{C}^Q_{s_1,\ldots,s_{i-1}|a_i|s_{i+1},\ldots,s_k} := \mathbb{C}\langle s_1\rangle_{a_1} \ldots \langle s_{i-1}\rangle_{a_{i-1}} \langle\langle\cdot\rangle\rangle_{a_i} \langle s_{i+1}\rangle_{a_{i+1}} \ldots \langle s_k\rangle_{a_k}$$

is a good context for $i \in \{1,\ldots,k\}$ and any terms $s_1,\ldots,s_{i-1},s_{i+1},\ldots,s_k$ such that $\mathbb{C}^Q_{s_1,\ldots,s_{i-1}|a_i|s_{i+1},\ldots,s_k}$ is a term.

Implementation Theorem. The well-bound and contextual decoding invariants are used to prove principal projection. The approximant and names invariants are used to prove the halt property. Search transparency and termination are straightforward. Then we apply the abstract implementation theorem (Theorem 13), and obtain the implementation theorem.

▶ **Proposition 19.** *Let $a \in \{\mathtt{axm}_1, \mathtt{axm}_2, \mathtt{axe}_1, \mathtt{axe}_2, \multimap, !\}$.*

1. Search transparency*: if $Q \rightsquigarrow_{sea} Q'$ then $\underline{Q} = \underline{Q'}$.*
2. Principal projection*: if $Q \rightsquigarrow_a Q'$ then $\underline{Q} \to_{\mathtt{G}a} \underline{Q'}$.*
3. Search termination*: transition $\rightsquigarrow_{sea}$ terminates.*
4. Halt*: if Q is final then $Q = (t \mid \epsilon)$ and $\underline{Q} = t$ is cut-free up to garbage.*

▶ **Theorem 20** (SESAME is good). *SESAME is a big-step implementation of the non-erasing good strategy $\to_{\mathtt{G}_{\neg\mathtt{w}}}$ of ESC.*

Complexity Analysis

The complexity analysis of SESAME amounts to bound the cost of implementing a run $r : t \triangleleft Q \rightsquigarrow^*_{\mathtt{SESAME}} Q'$ on random access machines (RAMs) as a function of two parameters: the number $|r|_{\mathtt{pr}}$ of principal transitions in r (which are in bijection with the number of $\to_{\mathtt{G}_{\neg\mathtt{w}}}$ steps) and the size $|t|$ of the initial term/state t/Q.

Sub-Term Property. The analysis relies on the *sub-term property*, ensuring that the duplicating principal transitions $\rightsquigarrow_!$ and $\rightsquigarrow_{\mathtt{axe}_1}$ manipulate only sub-terms of the initial term. Therefore, the cost of duplications is connected to size of the initial term. The property for SESAME can be inferred by the one for the good strategy, but we prefer to give a direct simple proof. The statement of the related invariant is about cut values, not duplications, but note that such values are the only terms duplicated by SESAME.

▶ **Lemma 21** (Sub-term). *Let $r\colon t \triangleleft Q \rightsquigarrow^*_{\mathtt{SESAME}} (\mathbb{C} \mid P)$ be a SESAME run.*

1. Invariant*: if v is a cut value in $\mathbb{C}$ or a value in P then $|v| \leq |t|$.*
2. Property*: if v is a value duplicated along r (by $\rightsquigarrow_!$ or $\rightsquigarrow_{\mathtt{axe}_1}$) then $|v| \leq |t|$.*

Since search transitions decrease the number of term constructors in the pool, which is only increased by principal transitions and of a quantity bounded by $|t|$ (by the sub-term invariant), we obtain the following bound.

▶ **Lemma 22** (Search transitions are bi-linear in number). *Let $r\colon t \triangleleft Q \rightsquigarrow^*_{\mathtt{SESAME}} Q'$ be a SESAME run. Then $|r|_{\mathtt{sea}} \leq |t| \cdot (|r|_{\mathtt{pr}} + 1)$.*

Cost of Single Transitions. For bounding the total cost, we need to make some hypotheses on how SESAME is going to be itself implemented on RAMs:

1. *Variable occurrences, binders, and cuts*: a variable is a memory location, a variable occurrence is a reference to it, and a cut $[v{\to}x]$ is the fact that the location associated with x contains v;
2. *Random access to variables*: the cuts in $\mathbb{C}$ can be accessed in $\mathcal{O}(1)$ by just following the reference given by the variable occurrence, with no need to search through $\mathbb{C}$;
3. *Named holes and jobs*: a named hole $\langle\cdot\rangle_a$ is again a memory location a, and a job $\langle t\rangle_a$ is a pointer to a and expresses the fact that location a contains t;
4. *Sequences of left rules*: cuts have a back pointer to the constructors before them (for instance in $[e?m][v{\to}x][m{\otimes}v', y]t$ the cut $[v{\to}x]$ has a pointer to $[e?m]$) so that the removal of a cut from the multi context $\mathbb{C}$ of a state – needed for the multiplicative transitions $\rightsquigarrow_{\mathtt{axm}_1}$, $\rightsquigarrow_{\mathtt{axm}_2}$, and $\rightsquigarrow_{\multimap}$ – can be performed in $\mathcal{O}(1)$.

As it is standard for time analyses, we also assume that pointers can be managed in $\mathcal{O}(1)$. These hypotheses mimic similar ones behind machines for λ-calculi, which are shown to be implementable in OCaml by Accattoli and Barras [8] and Accattoli et al. [12], and are followed by the OCaml implementation outlines in Sect. 9. They allow us to consider search transitions as having constant cost, and principal transitions as having cost bound by the initial term, by the sub-term property.

▶ **Lemma 23** (Cost of single transitions). *Let $r\colon t \triangleleft Q \rightsquigarrow^*_{\mathtt{SESAME}} Q'$ be a SESAME run. Search (resp. principal) transitions of r are implementable in $\mathcal{O}(1)$ (resp. $\mathcal{O}(|t|)$).*

Summing Up. By putting together the bounds on the number of search transitions with the cost of single transitions we obtain the complexity of SESAME.

▶ **Theorem 24** (SESAME bi-linear overhead bound). *Let $r\colon t \triangleleft Q \rightsquigarrow^*_{\mathtt{SESAME}} Q'$ be a SESAME run. Then r is implementable on RAMs in $\mathcal{O}\left(|t| \cdot (|r|_{\mathtt{pr}} + 1)\right)$.*

8 Final Garbage Collection and Full Cut Elimination

By the halt property (Proposition 19.4), SESAME stops on final state $Q = (\mathbb{A} \mid \epsilon)$ where $\mathbb{A}$ is a (pre-)term t that is cut-free *up to garbage*. The final garbage collection process turning t into a cut-free proof term $\mathtt{GC}(t)$ is formalized as the following function:

Final garbage Collection

$$\begin{array}{lll} \mathtt{GC}(x) := x & \mathtt{GC}(\lambda x.t) := \lambda x.\mathtt{GC}(t) & \mathtt{GC}(!t) := !\mathtt{GC}(t) \\ \mathtt{GC}([v{\to}x]t) := \mathtt{GC}(t) & \mathtt{GC}([m{\otimes}v, x]t) := [m{\otimes}\mathtt{GC}(v), x]\mathtt{GC}(t) & \mathtt{GC}([e?x]t) := [e?x]\mathtt{GC}(t) \end{array}$$

Clearly, $\mathtt{GC}(t)$ is cut-free. The following proposition states the qualitative and quantitative properties of garbage collection. Its first point uses the lemma before it, while the second point rests on the SESAME bi-linear bound.

▶ **Lemma 25.** *Let t be a term that is cut-free up to garbage but not cut-free. Then in t there is at least an out cut that is a $\to_{\mathtt{Gw}}$-redex.*

▶ **Proposition 26.** *Let $r : t \triangleleft Q \rightsquigarrow^*_{\mathtt{SESAME}} (s \mid \epsilon)$ a SESAME run ending on a final state.*

1. Good weakening steps simulate GC*: $s \to^k_{\mathtt{Gw}} \mathtt{GC}(s)$ where k is the number of out cuts in s;*
2. GC is bi-linear*: the final garbage collection function $s \mapsto \mathtt{GC}(s)$ can be implemented on RAMs in $\mathcal{O}(|t| \cdot (|r|_{\mathtt{pr}} + 1))$.*

By simulating good cut-elimination via the SESAME followed by final garbage collection, we obtain our main result, which – beyond the bi-linear bound – is a further slight (but sort of obvious) refinement of Accattoli's result, as it allows one to count only *non-erasing* good steps, rather than general good steps.

▶ **Theorem 27** (Good cut-elimination is big-step implementable with bi-linear overhead)**.** *Let $e : t \to_{\mathtt{G}}^{*} s$ be a good evaluation sequence with s cut-free. Then s is computable from t on RAMs in $\mathcal{O}(|t| \cdot (|e|_{\neg \mathtt{w}} + 1))$.*

Proof. If $e : t \to_{\mathtt{G}}^{*} s$ with s cut-free then $\to_{\mathtt{G}_{\neg \mathtt{w}}}$ terminates, by the *uniform normalization* corollary of the diamond property for $\to_{\mathtt{G}}$ (see the end of Sect. 3). Then there is an evaluation $e' : t \to_{\mathtt{G}_{\neg \mathtt{w}}}^{*} s'$ with s' cut-free up to garbage. Since SESAME is a big-step implementation of $\to_{\mathtt{G}_{\neg \mathtt{w}}}$ (Theorem 20), we obtain a SESAME run $r : t \triangleleft Q \rightsquigarrow_{\mathtt{SESAME}}^{*} Q'$ with $\underline{Q'} = s'$, of cost $\mathcal{O}\left(|t| \cdot (|r|_{\mathtt{pr}} + 1)\right) = \mathcal{O}(|t| \cdot (|e'| + 1))$. By the halt property of SESAME and cut-freeness up to garbage of SESAME, we obtain that Q' is final, that is, $Q' = (u \mid \epsilon)$ with $u =_{\alpha} s'$.

Since good weakening steps simulate GC (Proposition 26.1), we obtain $e'' : u \to_{\mathtt{Gw}}^{*} \mathtt{GC}(s')$ with $\mathtt{GC}(s')$ cut-free. Concatenating e' and e'' we obtain a good and normalizing evaluation sequence $e''' : t \to_{\mathtt{G}_{\neg \mathtt{w}}}^{*} \to_{\mathtt{Gw}}^{*} \mathtt{GC}(s')$. The diamond property of the good strategy (Theorem 10.2) implies *confluence* and *length invariance* of $\to_{\mathtt{G}}$ (see the end of Sect. 3), that is, we obtain $s = \mathtt{GC}(s')$ and $|e| = |e'''|$. Since computing GC is bi-linear (Proposition 26.1), and precisely costs again $\mathcal{O}\left(|t| \cdot (|r|_{\mathtt{pr}} + 1)\right) = \mathcal{O}(|t| \cdot (|e'| + 1))$, we obtain that the total cost of SESAME followed by GC is $2 \cdot \mathcal{O}(|t| \cdot (|e'| + 1)) = \mathcal{O}(|t| \cdot (|e'| + 1))$.

Lastly, the diamond property of the good strategy (Theorem 10.2) as stated in [4] does not give information about the kinds of steps, but by looking at its proof it is easily seen that diamond diagrams preserve the kind of steps. Therefore, not only we have $|e| = |e'''|$, but also $|e|_{\neg \mathtt{w}} = |e'''|_{\neg \mathtt{w}}$. Finally, note that $|e'''|_{\neg \mathtt{w}} = |e'|$, so the cost actually is $\mathcal{O}(|t| \cdot (|e|_{\neg \mathtt{w}} + 1))$. ◀

9 Implementation in OCaml

An implementation in OCaml can be found on GitHub at `https://github.com/sacerdot/sesame/`. All the datatypes and functions of the implementation are documented at `https://sacerdot.github.io/sesame`.

Aim and Design. The implementation is provided to support evidence for the cost of transitions claimed by Lemma 23. Moreover, it allows one to easily study the computational behaviour of family of terms by observing their evaluation.

The implementation has *not* been heavily optimized, in order to keep the code readable and close to the pen-and-paper presentation. Nevertheless, the employed data structures are reasonably close to those of an optimized implementation. Advanced recent OCaml features, like Generalized Algebraic Data Types, could have been used to statically enforce more invariants. We sticked instead to a simpler subset of OCaml, to make the code readable to non experts of the language. As a consequence, in several places there are assertions to abort the program in case the invariants are violated (possible only in case of bugs).

Overview. The implementation consists of a Read-Eval-Print-Loop (REPL) that asks the user to enter an IMELL term to be reduced, in ESC syntax. The accepted BNF is printed by the executable before starting the REPL. The term is first checked to be proper, then its strong normal form up to garbage is computed by the SESAME machine and finally garbage is removed. All intermediate machine steps are shown.

Before starting the REPL, a few test terms are reduced and printed (among which the example at the end of Sect. 6), to check the machine functionalities and as examples of the input syntax. In particular, the last tests are terms from the exploding family discussed in the next section.

Data Structures. Terms are encoded at runtime as term graphs, which are Direct Acyclic Graphs (DAG) augmented with a few back edges. Nodes in the graph are given by constructors of Algebraic Data Types whose arguments are mutable, to imperatively change the graph during reduction. All type declarations are given in the `termGraphs.ml` file. The data structures used in the implementation are described in Appendix H of the tech report [11].

Machine Runs: Auxiliary Functions and Their Complexity. The code that implements the runs of SESAME to normal form and the final garbage collection process can be found in `reduction.ml`, which is independent from all the remaining files but `termGraphs.ml`. Both files together amount to 412 lines of commented OCaml code.

A quick inspection of the code in `reduction.ml` shows only a few non-trivial functions, all the others being constant time:

1. `copy_{term,value,var,bvar}` that copies the DAG in input while visiting it. A $\mathcal{O}(1)$ fresh name generator is used to assign a new `name` value to copies of variables.
2. `alpha` and `enter_bo`, used respectively in the ! and $\multimap$ transitions, that take the body of a promotion/abstraction, traverse it to split it into a left context and a value, and build a new term by glueing together the obtained left context, a new cut that uses the value, and a remaining term. The two functions differ only in the fact that `alpha` also copies (α-renames) the term in input, calling the `copy_*` functions on the sub-terms, while `enter_bo` consumes the given term.
3. `steps`, the SESAME main loop, that runs until the pool is empty and a normal form is therefore reached, or it diverges otherwise.
4. `gc_{value,term}` that traverse in linear time the input to remove all garbage cuts from a normal form.

Linearity for all the previous functions but `steps` is easily established observing that the functions are based on visits of DAGs that never visit a node twice. In particular, the functions of Point 2 are linear in the size of the given term, which, by the sub-term invariant, is a (copy of a) sub-term of the initial term.

The only major source of technicalities – which however do not affect the complexity – is the fact that multiplicative transitions remove the acting cut from the multi context. For example, consider a multiplicative step $[e?f][v{\to}m][v'{\to}y]t \to [e?f][v'{\to}y]t'$ involving the cut on m. To implement it, beyond manipulating v, t, and t', one also has to connect $[e?f]$ and $[v'{\to}y]$. For that, cuts have a back-pointer to the preceding constructor (as mentioned at Point 4, labeled *sequences of left rule*, before Lemma 23), which induces inelegant imperative manipulations of the term graph, in particular in the copy function. The back-pointer on $[v{\to}m]$ is used to retrieve the dereliction $[e?f]$ from $[v{\to}m]$, while making the new connection of $[e?f]$ and $[v'{\to}y]$ requires changing two pointers: the one from the dereliction to its body, which has to target $[v'{\to}y]$, plus the back-pointer from $[v'{\to}y]$, which now has to target $[e?f]$.

The general scheme for addressing this issue is to augment the input t of functions that take a term with the pointer to the parent node of t, in order to be able to reassign the back-pointer when t is a cut.

Parsing, Pretty Printing, and Check for Properness. We have not attempted to achieve the best asymptotic costs for these functionalities since they are not relevant for the paper and since in practice we are manipulating small terms in input. To keep the code reproducible in the long term, we have not used any external library or tool for parsing and pretty-printing. The printed version of the example run at the end of Sect. 6 is shown in Appendix I of the tech report [11].

10 What's Next: an Interesting Family of Terms

We here hint at what we consider the most interesting future work, already mentioned by Accattoli in the conclusions of [4]. The difference is that here we provide a family of terms showing that such a future work is challenging.

The Question. *One* β-step of the λ-calculus is simulated in IMELL by *one* multiplicative step (actually a $\to_{\_\circ}$ step) followed by possibly *many* exponential steps. Since for many strategies the number of β-steps is a polynomial cost model, it means that one can count only the number of multiplicative (or even $\to_{\_\circ}$) steps, that is, *one can count zero for exponential steps*. Such a surprising fact is established easily in the case of weak evaluation with closed terms (roughly, it can be proved via standard abstract machines), while for strong evaluation it requires a sophisticated additional technique called *useful sharing* [15, 12, 20]. A question naturally arises: *is the number of ESC multiplicative/*$\to_{\_\circ}$ *good steps a polynomial time cost model as well?* This question – left to future work – is far from obvious. In the λ-calculus, there is a strong, hardcoded correlation between multiplicatives and exponentials, not present in IMELL. In the standard call-by-name/value encodings of λ-calculus in IMELL, indeed, multiplicatives and exponentials connectives *rigidly alternate*, while IMELL also has consecutive exponentials, as in $!!A$, enabling wilder exponential behaviour. A hint that that question might have a positive answer is given by the strong normalization of untyped exponentials in ESC/IMELL shown in [4] (while they are *not* SN in MELL, see [4]).

The Question is Challenging. We here show that the question mentioned above is challenging, and – surprisingly – it is already challenging in the apparently simple setting of basic evaluation with closed terms, that is the ESC analogous of weak evaluation for the λ-calculus, for which instead the question has an easy answer. The challenging aspect is here shown by building a family of ESC terms σ_n having size linear in n, whose evaluation is basic and made out of exponential steps only, and of a number of steps that is exponential in n. This is impossible in the λ-calculus, and it is crucially related to iterated exponential terms such as $!!t$. Since the evaluation of σ_n uses 0 multiplicative steps, it suggests that the number of multiplicative steps is *not* a polynomial cost model. More precisely, it shows that a smart additional mechanism – akin to useful sharing – is needed in order to circumvent the exponential number of exponential steps. Useful sharing as it appears in the literature, however, cannot be the answer, since useful sharing addresses exponential inefficiencies that are induced by strong evaluation, which is an orthogonal issue.

The Exponential Exploding Family. The family is built in three steps.

1. For $k \geq 1$, define $\pi_k := \overbrace{[f?\_]\dots[f?\_][f?e]}^{k} e$, where $\_$ is an exponential variable with no occurrences, and whose name is irrelevant. These terms can be composed as to reduce to (the cut-free term) $\pi_{k\cdot h}$ via exponential basic evaluation, that is, $[!\pi_k \to f]\pi_h \to_{\mathsf{e}}^* \pi_{k\cdot h}$.

2. Define $\delta_n := \overbrace{[!\pi_2 \to f] \dots [!\pi_2 \to f]}^{n-1} \pi_2$, and note that, by Point 1, one has $\delta_n \to_{\mathsf{e}}^* \pi_{2^n}$.
3. Finally, define the exponential exploding family as $\sigma_n := [!!\lambda m.m \to f]\delta_n$. By Point 2, we have $\sigma_n \to_{\mathsf{e}}^* [!!\lambda m.m \to f]\pi_{2^n}$. Then note that $[!!\lambda m.m \to f]\pi_{2^n} \to_{\mathsf{e}}^{\Omega(2^n)} !\lambda m.m$.

Summing up, σ_n is an example of term of size $\mathcal{O}(n)$ that reduces in $\Omega(2^n)$ exponential steps and 0 multiplicative steps to a cut-free proof using only basic evaluation. As mentioned in the previous section, the OCaml implementation starts by running some tests, including examples of the three points above, respectively for $k = 3, h = 4$, for $n = 3$, and for $n = 3$.

References

1 Samson Abramsky. Computational interpretations of linear logic. *Theor. Comput. Sci.*, 111(1&2):3–57, 1993. doi:10.1016/0304-3975(93)90181-R.

2 Beniamino Accattoli. An abstract factorization theorem for explicit substitutions. In Ashish Tiwari, editor, *23rd International Conference on Rewriting Techniques and Applications (RTA'12) , RTA 2012, May 28 - June 2, 2012, Nagoya, Japan*, volume 15 of *LIPIcs*, pages 6–21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPICS.RTA.2012.6.

3 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*, pages 1–21, 2016. doi:10.1007/978-3-662-52921-8_1.

4 Beniamino Accattoli. Exponentials as substitutions and the cost of cut elimination in linear logic. *Log. Methods Comput. Sci.*, 19(4), 2023. doi:10.46298/LMCS-19(4:23)2023.

5 Beniamino Accattoli and Pablo Barenbaum. A diamond machine for strong evaluation. In Chung-Kil Hur, editor, *Programming Languages and Systems - 21st Asian Symposium, APLAS 2023, Taipei, Taiwan, November 26-29, 2023, Proceedings*, volume 14405 of *Lecture Notes in Computer Science*, pages 69–90. Springer, 2023. doi:10.1007/978-981-99-8311-7_4.

6 Beniamino Accattoli, Pablo Barenbaum, and Damiano Mazza. Distilling abstract machines. In Johan Jeuring and Manuel M. T. Chakravarty, editors, *Proceedings of the 19th ACM SIGPLAN international conference on Functional programming, Gothenburg, Sweden, September 1-3, 2014*, pages 363–376. ACM, 2014. doi:10.1145/2628136.2628154.

7 Beniamino Accattoli, Pablo Barenbaum, and Damiano Mazza. A strong distillery. *CoRR*, abs/1509.00996, 2015. arXiv:1509.00996.

8 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, Namur, Belgium, October 09 - 11, 2017*, pages 4–16. ACM, 2017. doi:10.1145/3131851.3131855.

9 Beniamino Accattoli, Eduardo Bonelli, Delia Kesner, and Carlos Lombardi. A nonstandard standardization theorem. In Suresh Jagannathan and Peter Sewell, editors, *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 659–670. ACM, 2014. doi:10.1145/2535838.2535886.

10 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.

11 Beniamino Accattoli and Claudio Sacerdoti Coen. IMELL Cut Elimination with Linear Overhead, 2024. arXiv:2405.03669.

12 Beniamino Accattoli, Andrea Condoluci, and Claudio Sacerdoti Coen. Strong call-by-value is reasonable, implosively. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–14. IEEE, 2021. doi:10.1109/LICS52264.2021.9470630.

13 Beniamino Accattoli, Andrea Condoluci, Giulio Guerrieri, and Claudio Sacerdoti Coen. Crumbling abstract machines. In *Proceedings of the 21st International Symposium on Principles and Practice of Programming Languages, PPDP 2019, Porto, Portugal, October 7-9, 2019*, pages 4:1–4:15. ACM, 2019. doi:10.1145/3354166.3354169.

14 Beniamino Accattoli and Ugo Dal Lago. On the Invariance of the Unitary Cost Model for Head Reduction. In *23rd International Conference on Rewriting Techniques and Applications (RTA'12) , RTA 2012, May 28 - June 2, 2012, Nagoya, Japan*, pages 22–37. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.RTA.2012.22.

15 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.

16 Beniamino Accattoli, Ugo Dal Lago, and Gabriele Vanoni. The (in)efficiency of interaction. *Proc. ACM Program. Lang.*, 5(POPL):1–33, 2021. doi:10.1145/3434332.

17 Beniamino Accattoli, Ugo Dal Lago, and Gabriele Vanoni. The space of interaction. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–13. IEEE, 2021. doi:10.1109/LICS52264.2021.9470726.

18 Beniamino Accattoli and Giulio Guerrieri. Abstract machines for open call-by-value. *Sci. Comput. Program.*, 184, 2019. doi:10.1016/j.scico.2019.03.002.

19 Beniamino Accattoli, Ugo Dal Lago, and Gabriele Vanoni. Reasonable space for the λ-calculus, logarithmically. In Christel Baier and Dana Fisman, editors, *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2 - 5, 2022*, pages 47:1–47:13. ACM, 2022. doi:10.1145/3531130.3533362.

20 Beniamino Accattoli and Maico Leberle. Useful open call-by-need. In Florin Manea and Alex Simpson, editors, *30th EACSL Annual Conference on Computer Science Logic, CSL 2022, February 14-19, 2022, Göttingen, Germany (Virtual Conference)*, volume 216 of *LIPIcs*, pages 4:1–4:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.CSL.2022.4.

21 Francisco Alberti and Eike Ritter. An efficient linear abstract machine with single-pointer property, 1998. ESSLLI.

22 Malgorzata Biernacka, Witold Charatonik, and Tomasz Drab. A derived reasonable abstract machine for strong call by value. In Niccolò Veltri, Nick Benton, and Silvia Ghilezan, editors, *PPDP 2021: 23rd International Symposium on Principles and Practice of Declarative Programming, Tallinn, Estonia, September 6-8, 2021*, pages 6:1–6:14. ACM, 2021. doi:10.1145/3479394.3479401.

23 Malgorzata Biernacka, Witold Charatonik, and Tomasz Drab. A simple and efficient implementation of strong call by need by an abstract machine. *Proc. ACM Program. Lang.*, 6(ICFP):109–136, 2022. doi:10.1145/3549822.

24 Guy E. Blelloch and John Greiner. Parallelism in sequential functional languages. In *Proceedings of the seventh international conference on Functional programming languages and computer architecture, FPCA 1995, La Jolla, California, USA, June 25-28, 1995*, pages 226–237, 1995. doi:10.1145/224164.224210.

25 Eduardo Bonelli. The linear logical abstract machine. In Stephen D. Brookes and Michael W. Mislove, editors, *Proceedings of the 22nd Annual Conference on Mathematical Foundations of Programming Semantics, MFPS 2006, Genova, Italy, May 23-27, 2006*, volume 158 of *Electronic Notes in Theoretical Computer Science*, pages 99–121. Elsevier, 2006. doi:10.1016/J.ENTCS.2006.04.007.

26 Pierre Crégut. Strongly reducing variants of the Krivine abstract machine. *High. Order Symb. Comput.*, 20(3):209–230, 2007. doi:10.1007/s10990-007-9015-z.

27 Ugo Dal Lago and Simone Martini. An invariant cost model for the lambda calculus. In Arnold Beckmann, Ulrich Berger, Benedikt Löwe, and John V. Tucker, editors, *Logical Approaches to Computational Barriers, Second Conference on Computability in Europe, CiE 2006, Swansea, UK, June 30-July 5, 2006, Proceedings*, volume 3988 of *Lecture Notes in Computer Science*, pages 105–114. Springer, 2006. doi:10.1007/11780342_11.

28 Ugo Dal Lago and Simone Martini. Derivational complexity is an invariant cost model. In Marko C. J. D. van Eekelen and Olha Shkaravska, editors, *Foundational and Practical Aspects of Resource Analysis - First International Workshop, FOPARA 2009, Eindhoven, The Netherlands, November 6, 2009, Revised Selected Papers*, volume 6324 of *Lecture Notes in Computer Science*, pages 100–113. Springer, 2009. doi:10.1007/978-3-642-15331-0_7.

29 Ugo Dal Lago and Simone Martini. On constructor rewrite systems and the lambda-calculus. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming, 36th Internatilonal Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part II*, volume 5556 of *Lecture Notes in Computer Science*, pages 163–174. Springer, 2009. doi:10.1007/978-3-642-02930-1_14.

30 Vincent Danos and Laurent Regnier. Reversible, irreversible and optimal lambda-machines. In Jean-Yves Girard, Mitsuhiro Okada, and Andre Scedrov, editors, *Linear Logic Tokyo Meeting 1996, Keio University, Mita Campus, Tokyo, Japan, March 29 - April 2, 1996*, volume 3 of *Electronic Notes in Theoretical Computer Science*, pages 40–60. Elsevier, 1996. doi:10.1016/S1571-0661(05)80402-5.

31 Hugo Herbelin. A lambda-calculus structure isomorphic to gentzen-style sequent calculus structure. In Leszek Pacholski and Jerzy Tiuryn, editors, *Computer Science Logic, 8th International Workshop, CSL '94, Kazimierz, Poland, September 25-30, 1994, Selected Papers*, volume 933 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 1994. doi:10.1007/BFB0022247.

32 Delia Kesner and Shane Ó Conchúir. Milner's lambda-calculus with partial substitutions. *CoRR*, abs/2312.13270, 2023. arXiv:2312.13270, doi:10.48550/arXiv.2312.13270.

33 Yves Lafont. The linear abstract machine. *Theor. Comput. Sci.*, 59:157–180, 1988. doi:10.1016/0304-3975(88)90100-4.

34 Ugo Dal Lago and Simone Martini. The weak lambda calculus as a reasonable machine. *Theor. Comput. Sci.*, 398(1-3):32–50, 2008. doi:10.1016/J.TCS.2008.01.044.

35 Ian Mackie. The geometry of interaction machine. In Ron K. Cytron and Peter Lee, editors, *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. ACM Press, 1995. doi:10.1145/199448.199483.

36 Ian Mackie. Interaction nets for linear logic. *Theor. Comput. Sci.*, 247(1-2):83–140, 2000. doi:10.1016/S0304-3975(00)00198-5.

37 Ian Mackie and Jorge Sousa Pinto. Encoding linear logic with interaction combinators. *Inf. Comput.*, 176(2):153–186, 2002. doi:10.1006/INCO.2002.3163.

38 Ian Mackie and Shinya Sato. A calculus for interaction nets based on the linear chemical abstract machine. In Vincent Danos and Mariangiola Dezani, editors, *Proceedings of the Third International Workshop on Developments in Computational Models, DCM@ICALP 2007, Wroclaw, Poland, July 15, 2007*, volume 192 of *Electronic Notes in Theoretical Computer Science*, pages 59–70. Elsevier, 2007. doi:10.1016/J.ENTCS.2008.10.027.

39 Seikoh Mikami and Yohji Akama. A study of abramsky's linear chemical abstract machine. In Jean-Yves Girard, editor, *Typed Lambda Calculi and Applications, 4th International Conference, TLCA'99, L'Aquila, Italy, April 7-9, 1999, Proceedings*, volume 1581 of *Lecture Notes in Computer Science*, pages 243–257. Springer, 1999. doi:10.1007/3-540-48959-2_18.

40 Robin Milner. Local bigraphs and confluence: Two conjectures (extended abstract). In Roberto M. Amadio and Iain Phillips, editors, *Proceedings of the 13th International Workshop on Expressiveness in Concurrency, EXPRESS 2006, Bonn, Germany, August 26, 2006*, volume 175 of *Electronic Notes in Theoretical Computer Science*, pages 65–73. Elsevier, 2006. doi:10.1016/J.ENTCS.2006.07.035.

41 Claudio Sacerdoti Coen. sesame. Software, swhId: swh:1:dir:93776e14435d0dd5cde6deee3d9cf8f515f18fac (visited on 2024-06-19). URL: https://github.com/sacerdot/sesame/.

42 David Sands, Jörgen Gustavsson, and Andrew Moran. Lambda calculi and linear speedups. In Torben Æ. Mogensen, David A. Schmidt, and Ivan Hal Sudborough, editors, *The Essence of Computation, Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones [on occasion of his 60th birthday]*, volume 2566 of *Lecture Notes in Computer Science*, pages 60–84. Springer, 2002. doi:10.1007/3-540-36377-7_4.

43 Shinya Sato, Toru Sugimoto, and Shinichi Yamada. An implementation model of the typed lambda-calculus based on linear chemical abstract machine. In Michael Hanus, editor, *International Workshop on Functional and (Constraint) Logic Programming, WFLP 2001, Kiel, Germany, September 13-15, 2001, Selected Papers*, volume 64 of *Electronic Notes in Theoretical Computer Science*, pages 292–307. Elsevier, 2001. doi:10.1016/S1571-0661(04)80356-6.

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

45 David N. Turner and Philip Wadler. Operational interpretations of linear logic. *Theor. Comput. Sci.*, 227(1-2):231–248, 1999. doi:10.1016/S0304-3975(99)00054-7.

46 Gabriele Vanoni. *On Reasonable Space and Time Cost Models for the λ-Calculus. (Sur les modèles de coût raisonnable en espace et en temps pour le λ-calcul)*. PhD thesis, University of Bologna, Italy, 2022. URL: https://tel.archives-ouvertes.fr/tel-03923206.

Substitution for Non-Wellfounded Syntax with Binders Through Monoidal Categories

Ralph Matthes
IRIT, Université de Toulouse, CNRS, Toulouse INP, UT3, Toulouse, France

Kobe Wullaert
Delft University of Technology, The Netherlands

Benedikt Ahrens
Delft University of Technology, The Netherlands
University of Birmingham, United Kingdom

Abstract

We describe a generic construction of non-wellfounded syntax involving variable binding and its monadic substitution operation.

Our construction of the syntax and its substitution takes place in category theory, notably by using monoidal categories and strong functors between them. A language is specified by a multi-sorted binding signature, say Σ. First, we provide sufficient criteria for Σ to generate a language of possibly infinite terms, through ω-continuity. Second, we construct a monadic substitution operation for the language generated by Σ. A cornerstone in this construction is a mild generalization of the notion of heterogeneous substitution systems developed by Matthes and Uustalu; such a system encapsulates the necessary corecursion scheme for implementing substitution.

The results are formalized in the Coq proof assistant, through the UniMath library of univalent mathematics.

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

Keywords and phrases Non-wellfounded syntax, Substitution, Monoidal categories, Actegories, Tensorial strength, Proof assistant Coq, UniMath library

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.25

Related Version *Full Version*: `https://arxiv.org/abs/2308.05485` [23]

Acknowledgements We thank Henning Basold for pointing us to the work on completely iterative algebras, leading to a simpler proof of Theorem 4. We also thank Thomas Lamiaux for valuable comments on a draft of this paper. We gratefully acknowledge the work by the Coq development team in providing the Coq proof assistant and surrounding infrastructure, as well as their support in keeping UniMath compatible with Coq. Not least, we thank the anonymous FSCD reviewers for their thoughtful feedback on our submission.

1 Introduction

1.1 General Motivation for Non-Wellfounded Syntax With Binders

Non-wellfounded syntax with binders appears in its purest form in the coinductive reading of untyped λ-calculus. *Potentially* non-wellfounded λ-terms still consist of variables, λ-abstractions and applications only, but the construction process with these constructors can go on forever. Such construction processes can be described through functional programming, and the host programming language then serves as a meta-language for the description of those infinitary λ-terms. Instead of taking a programming perspective, one can also ask if a possibly circular definition of such a non-wellfounded term is well-formed, in the sense

that it uniquely determines such a structure. Naturally, uniqueness is understood up to bisimilarity, i. e., two such non-wellfounded λ-terms are considered equal if their infinite unfoldings have the same labels (indicating the applied constructor) in the same order on each level, starting at the root. The presence of variable binding presents the extra challenge of having to consider this bisimilarity modulo renaming of bound variables, i. e., α-equivalence – a challenge that is amplified by the possibility of having an infinite number of bindings in a non-wellfounded λ-term. In this paper, we either work on an abstract level that does not reveal this challenge, or we resort to a representation using nested datatypes that is a form of de Bruijn representation with well-scopedness guaranteed by the typing system (see, e. g., [6]), and therefore α-equivalence is just not needed.

There are many uses of coinductive untyped λ-terms (such as Böhm trees), and coinductive readings of term structures with binding (also with simple types, e. g., as in an automata-theoretic analysis by Melliès [24]) have a counterpart in infinitary rewriting.

1.2 A Motivational Application Scenario

We have an application scenario in mind for which the "static" part, i. e., the well-typed syntax itself, is important, even without the aforementioned "dynamics" of infinitary rewriting. It is more complicated than just λ-calculus, notably by the presence of embedded inductive types. This in particular motivates our search for *datatype-generic* constructions for a wide range of non-wellfounded simply-typed syntax.

The application scenario is as follows: We want to represent the entire search space for inhabitants in simply-typed λ-calculus (STLC) by a potentially non-wellfounded term of a suitable calculus. The inhabitation problem itself is the following question: "Given a context Γ and a type A of STLC, is there a term t of STLC such that $\Gamma \vdash t : A$?". Taking into account the entire search space means including infinite runs that arise in a (naive) search loop. And the term of the "suitable calculus" should again have type A in context Γ but represent the search space and not only be a single inhabitant. This calculus is informally given by the following grammar:

$$\begin{array}{lrll} \text{(terms)} & N & ::=_{co} & \lambda x^A.N \mid E_1 + \cdots + E_n \\ \text{(elimination alternatives)} & E & ::=_{co} & x\langle N_1, \ldots, N_k\rangle \end{array}$$

with one constructor for each $n, k \geq 0$, hence we have sums with any finite number of summands and tuples with any finite number of arguments. We write x in place of $x\langle\rangle$ – this captures $k = 0$. The elimination alternatives resemble the neutral terms of λ-calculus of the form $xN_1 \ldots N_k$ – we are only searching for inhabitants in normal form. They have this name because they correspond to repeated implication elimination (as expressed by STLC typing) and they are summands in $E_1 + \cdots + E_n$ that indicate a finite choice between those "alternative" n summands. Search for normal forms in STLC only has finitely many options at each choice point, even though, e. g., there are infinitely many inhabitants of the type of Church numerals.

The elements of the syntactic category of terms are also called "forests". The index co means that the grammar is read coinductively. There are two clauses that embed (finite) lists into the codata type. It therefore presents at least the challenges of non-wellfounded "rose trees", i. e., finitely-branching unlabeled trees without a bound on the branching width. The scenario comes from [27, Section 3.2], and we plan to study it with our formalization. The typing rules for these expressions are given in Figure 1, with Γ ranging over finite(!) typing contexts. They are the usual implication introduction and a vectorized implication elimination (down to atomic types p), and the rule for typing alternatives (of the same

$$\frac{\Gamma, x : A \vdash N : B}{\Gamma \vdash \lambda x^A.N : A \to B}co \qquad \frac{(x : \vec{B} \to p) \in \Gamma \quad \forall i \leq k, \Gamma \vdash N_i : B_i}{\Gamma \vdash x\langle N_i\rangle_{i\leq k} : p}co \qquad \frac{\forall i \leq n, \Gamma \vdash E_i : p}{\Gamma \vdash \sum_{i\leq n} E_i : p}co$$

Figure 1 Coinductive typing rules for simple types in the application scenario.

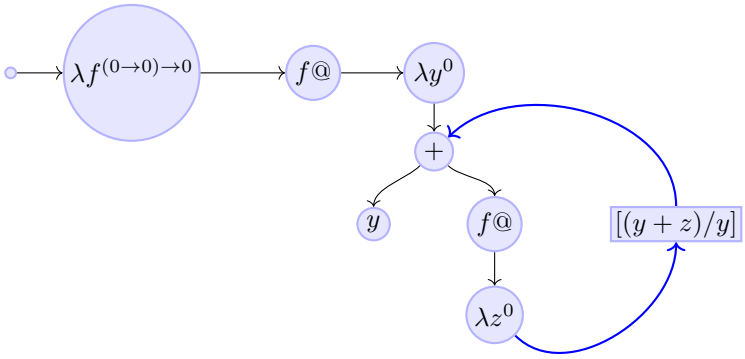


Figure 2 Forest representation of all inhabitants of THREE.

atomic type) – and all rules are read coinductively, indicated by the *co* mark. A well-typed such term hence *locally* conforms to intuitionistic implicational logic. For illustration, we give a well-typed forest in graphical form – the much easier example of Church numerals is found in Appendix A. Let $\mathsf{THREE} :\equiv ((0 \to 0) \to 0) \to 0$ for an atom 0. This is the simplest type of rank (i. e., nesting depth) 3. We define a closed forest of type THREE in Figure 2 [27, Example 16]. $f@$ is short for $f\langle N\rangle$ with N given by where the arrow points to. The "decontraction operation" in the back link resides on the meta-level and is specific to the summation in this example grammar: $[(y + z)/y]$ (written $[y : 0, z : 0/y : 0]$ in the cited paper) is decontraction and says that every occurrence of y has to be replaced by a sum once with y and once with z in place of the original y. This forest representation can be seen as a formal approach to the informal concept of "inhabitation machines" [8, pp. 34–38]. All the inhabitants of THREE can be read off this forest: omitting types, they are of the form $\lambda f.f\langle\lambda y_1.f\langle\lambda y_2.f\langle\cdots\langle\lambda y_n.y_i\rangle\cdots\rangle\rangle\rangle$, with $1 \leq i \leq n$. The individual inhabitants are wellfounded, but the forests representing the entire search spaces (for all simple types) are obtained coinductively in [27]. We use a generic construction of syntax such as the forests of this scenario that is based on category theory.

1.3 Context and Overview of this Paper

For *wellfounded* languages with variable binding, categorical semantics are given in [17]. The importance of monoidal structure for the modelling of substitution is emphasized there; many of the constructions are given on the level of monoidal categories, and are later instantiated to a suitable category of contexts. A very extensive overview of work on substitution for wellfounded syntax with binders, comparing [17] and subsequent work by the same and other authors, is given by Lamiaux and Ahrens [21]. A categorical semantics of *non*-wellfounded syntax with binding appeared in [22] involving the first author. That work is set concretely in endofunctor categories instead of general monoidal categories.

In that context, the present paper makes the following contributions. Firstly, the new definitions and results of the present paper lift the approach of [22] to the abstraction level of monoidal categories – to reach the same abstraction level as [17]. Secondly, we provide a full type-theoretic formalization of the results on the abstract level. Using both of these contributions, by (non-trivial) instantiation, we get a tool chain from multi-sorted binding signatures to certified monadic substitution for non-wellfounded syntax and thus the non-wellfounded counterpart to the tool chain described in [5] involving two of the present authors. Such a tool chain is absent from [22] even on the informal level (neither multi-sorted binding signatures nor unsorted binding signatures are considered). Our approach is now general enough so that the monoidal category underlying [17] and its ramifications can also be studied concerning non-wellfounded syntax and substitution for it.

In more detail, we construct non-wellfounded syntax from final coalgebras in suitable functor categories, hence based on category theory. Variable binding is modelled through the use of nested datatypes, as, e.g., in [10, 11]. The structure map of the final coalgebra is, of course, an isomorphism with its inverse providing an algebra structure. This excludes the existence of exotic terms. We benefit from that abstract view in order to construct a monadic substitution operation similar to [7], i. e., a meta-level operation that is not specific to the application scenario presented above (in contrast to the decontraction operation it features). The qualifier "monadic" implies that the generic approach includes proving the monad laws. For the case of wellfounded syntax, this is well-established in the literature (see, e. g., [7]). For the non-wellfounded case but without types, this has also been done before [20].

Further previewing our technical contributions, our approach is to generalize the notion of heterogeneous substitution systems [22] from endofunctor categories to monoidal categories. Those systems (abbreviated HSS) were meant as a tool to construct monadic substitution both for wellfounded and non-wellfounded syntax – by a common abstraction that serves as a pivotal structure between initial algebras and final coalgebras, respectively, on the input side and the substitution monad as output. We call the generalization *monoidal heterogeneous substitution systems* (MHSS). All these ingredients have been considered before for the sake of representation of wellfounded syntax [16, Section I.1.2][19, Section 5.2.1]. Section 3 is the core contribution of this paper, making the step to non-wellfounded syntax on the more abstract level of monoidal categories. The results in that section demonstrate the pivotal role of MHSS: from a final coalgebra, a MHSS is constructed, and from a MHSS, a monoid is constructed, which abstracts away from monadic substitution. Section 4 applies the results of Section 3 to the endofunctor scenario – which is hardwired into the definitions in [22].

1.4 Synopsis

The remainder of this paper is structured as follows. In Section 2 we review some prerequisites from category theory that are used later in the paper. Section 3 presents the construction of non-wellfounded syntax with substitution on the level of monoidal categories. As promised above, it generalizes both to monoidal categories and from wellfounded to non-wellfounded syntax – sloppily construable as "pushout" of these two directions. Section 4 applies the results of Section 3 to the endofunctor scenario, capturing simply-typed non-wellfounded syntax (with binding) generically. The appendix contains technical complements. However, for lack of space, Appendix B and Appendix D are only present in the full version [23].

All of the definitions and results presented in this paper (except for the motivational application scenario in Section 1.2) are formalized and computer-checked in `UniMath` [29], a library of univalent mathematics based on the computer proof assistant `Coq` [28]. Throughout this paper, definitions and results are annotated by `Coq` identifiers for the corresponding

definitions and results in our library. These identifiers are hyperlinks leading to an HTML version of the proof code; for instance, clicking on `monoidal` brings you to the definition of monoidal category. The formalization is not the main topic of this paper (a discussion of some formalization aspects is found in Appendix E); we use it mainly to relieve ourselves from the burden of writing out lengthy and uninteresting proofs, and the reader from the burden of reading them. Instead, we restrict ourselves to pointing the reader to interesting aspects of proofs and constructions, and aim to convey the intuition behind – and useful applications of – our work.

2 Preliminaries

We assume working knowledge of category theory and mostly only point to specific choices of notation. We write $a : \mathcal{C}$ to indicate that a is an object in category $\mathcal{C}$; we write $f : a \to b$ to indicate that f is a morphism from a to b in $\mathcal{C}$. Following the choice adopted for the `UniMath` library, composition is written in "diagrammatic" order, i. e., the composite of $f : a \to b$ and $g : b \to c$ is denoted $f \cdot g : a \to c$. We will make use of a category Set of sets; objects of this category are called "small" sets.

2.1 Monoidal Categories and Actegories

In this section we briefly review the notions of monoidal category and of actegory.

A monoidal category is given by a six-tuple $(\mathcal{C}, \otimes, I, \lambda, \rho, \alpha)$ where $\mathcal{C}$ is a category, $\otimes : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$, $I : \mathcal{C}$, $\lambda = (\lambda_x)_{x:\mathcal{C}}$ (the *left unitor*) with $\lambda_x : I \otimes x \to x$, $\rho = (\rho_x)_{x:\mathcal{C}}$ (the *right unitor*) with $\rho_x : x \otimes I \to x$ and $\alpha = (\alpha_{x,y,z})_{x,y,z:\mathcal{C}}$ (the *associator*) with $\alpha_{x,y,z} : (x \otimes y) \otimes z \to x \otimes (y \otimes z)$. The unitors and the associator are required to be natural isomorphisms, and are furthermore subject to coherence laws called the "triangle law" and the "pentagon law", recalled in Appendix B that is only available in the full version [23]. We will use the letter $\mathcal{V}$ to indicate the first component of a monoidal category and, by slight abuse of language, we even call $\mathcal{V}$ a monoidal category when the other components are left implicit. We can also just mention $(\mathcal{C}, \otimes, I)$ or $(\mathcal{V}, \otimes, I)$.

For the proper understanding of the strength notion and for the construction process of a strength in our application scenario, we use actions of monoidal categories on categories, called *actegories*. For the naming of concepts, we vaguely follow [14].

Given a monoidal category $\mathcal{V}$, a (left) $\mathcal{V}$-actegory is given by a quadruple $(\mathcal{C}, \odot, \lambda, \mathsf{act})$ where $\mathcal{C}$ is a category, $\odot : \mathcal{V} \times \mathcal{C} \to \mathcal{C}$ (the *action*), $\lambda = (\lambda_x)_{x:\mathcal{C}}$ (the *unitor*) with $\lambda_x : I \odot x \to x$, and $\mathsf{act} = (\mathsf{act}_{v,w,x})_{v,w:\mathcal{V},x:\mathcal{C}}$ (the *actor*) with $\mathsf{act}_{v,w,x} : (v \otimes w) \odot x \to v \odot (w \odot x)$. The unitor and the actor are required to be natural isomorphisms, and are furthermore required to satisfy coherence laws analogous to the ones for monoidal categories, called also "triangle law" and "pentagon law" that are found in Appendix B (only in [23]). We consider it as important that actegories are a kind of widening of the concept of monoidal categories, in the following sense: Given a monoidal category $(\mathcal{V}, \otimes, I, \lambda, \rho, \alpha)$, $(\mathcal{V}, \otimes, \lambda, \alpha)$ is a $\mathcal{V}$-actegory, and it is called the actegory *with the canonical self-action* (cf. `actegory_with_canonical_self_action`).

We furthermore consider *strong monoidal functors* from monoidal category $(\mathcal{C}, \otimes, I)$ to monoidal category $(\mathcal{D}, \otimes', I')$. Such a functor is given by a triple (F, ϵ, μ), where $F : \mathcal{C} \to \mathcal{D}$ (the *underlying functor*), $\epsilon : I' \to FI$ (preservation of unit) and $\mu = (\mu_{x,y})_{x,y:\mathcal{C}}$ (preservation of tensor) with $\mu_{x,y} : Fx \otimes' Fy \to F(x \otimes y)$. Here, we assume μ to be a natural transformation, and ϵ and μ to be isomorphisms (so as to be "strong"), as well as the three well-known (lax) laws of preservation of left and right unitality and associativity. By abuse of notation, we even call F a strong monoidal functor when the other components are left implicit.

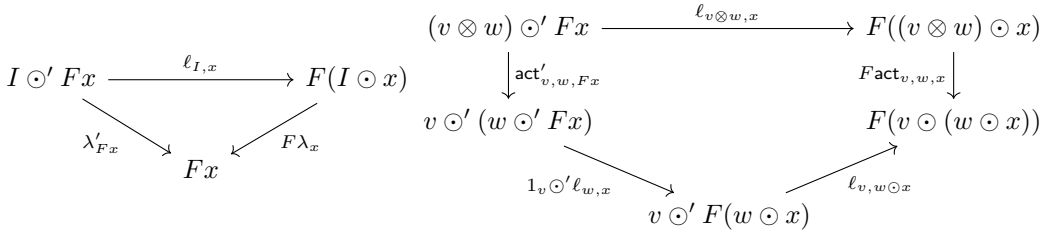


Figure 3 Preservation of the unitor and the actor in a linear functor.

For the purposes of syntax representation, we only consider the lax form of morphisms between actegories, over a common monoidal category $\mathcal{V}$. Given a monoidal category $(\mathcal{V}, \otimes, I)$, a lax linear functor from actegory $(\mathcal{C}, \odot, \lambda, \mathsf{act})$ to actegory $(\mathcal{D}, \odot', \lambda', \mathsf{act}')$ is given by a pair (F, ℓ), where $F : \mathcal{C} \to \mathcal{D}$ (the *underlying functor*) and $\ell = (\ell_{v,x})_{v:\mathcal{V},x:\mathcal{C}}$ (the *lineator*) with $\ell_{v,x} : v \odot' Fx \to F(v \odot x)$. We require the lineator to be a natural transformation (not necessarily an isomorphism), and furthermore require it to satisfy two laws of preservation of the unitor and the actor, see Figure 3. Currying away the second index to ℓ and using that λ and act are isomorphisms, these laws uniquely determine ℓ_I and give a formula to calculate $\ell_{v\otimes w}$ from ℓ_v and ℓ_w. (On this level of generality of the description, this is not different from the situation for the μ component of a strong monoidal functor.)

2.2 Pointed Strength

Pointed strength is best understood through actegories; this is sketched in [16, Section I.1.2]), and our presentation here has the same main ingredients (using reindexing and a coslice category, see below). This abstract view is helpful for creating libraries of functors with pointed strength, as will be visible in Section 4. Hur uses the notion of pointed strength extensively but only spells it out concretely [19, Section 5.2.1].

Given monoidal categories $\mathcal{W}$ and $\mathcal{V}$, a strong monoidal functor $F : \mathcal{W} \to \mathcal{V}$ and a $\mathcal{V}$-actegory $(\mathcal{C}, \odot, \lambda, \mathsf{act})$, one can canonically construct a $\mathcal{W}$-actegory $(\mathcal{C}, \odot', \lambda', \mathsf{act}')$ over the same base category – the *reindexing of the $\mathcal{V}$-actegory along F*. (It seems that it would suffice that F is an oplax monoidal functor instead of a strong one.) On objects, the action $\odot'$ is constructed as $w \odot' x :\equiv Fw \odot x$. We will not spell out the details here; our formalization of this construction is given in `reindexed_actegory`.

We need reindexed actegories for one specific situation: the actegory with the canonical pointed action. We assume a monoidal category $\mathcal{V}$ with unit I and construct the monoidal category of "monoidal-pointed objects": the underlying category is the coslice category $I/\mathcal{V}$ whose objects are pairs (v, pv) with $v : \mathcal{V}$ and $pv : I \to v$ ("a point for v"), and the monoidal category can be easily constructed. Just for the record: $I^{ptd} :\equiv (I, 1_I)$ is the unit, and the tensor is defined on objects as $(v, pv) \otimes^{ptd} (w, pw) :\equiv (v \otimes w, \lambda_I^{-1} \cdot (pv \otimes pw))$.

Given a monoidal category $\mathcal{V}$, the actegory $\mathcal{V}^{ptd}$ that we call the *actegory with the canonical pointed action of $\mathcal{V}$* is obtained by reindexing: in the definition above, we take $\mathcal{W} := I/\mathcal{V}$, $\mathcal{V}$ as given, F the forgetful functor that forgets the points (and is strong monoidal), and as $\mathcal{V}$-actegory the actegory with the canonical self-action of $\mathcal{V}$ introduced above. It follows that in $\mathcal{V}^{ptd}$, the monoidal-pointed objects of $\mathcal{V}$ act on the objects of $\mathcal{V}$.

Given a monoidal category $\mathcal{V}$ and an endofunctor F on $\mathcal{V}$, a *pointed tensorial strength* for F is a θ so that (F, θ) is a lax endomorphism of actegory $\mathcal{V}^{ptd}$. In other words, θ is the lineator (following the literature, we use θ for this specific use of lineators) in the situation where source and target action are $\mathcal{V}^{ptd}$.

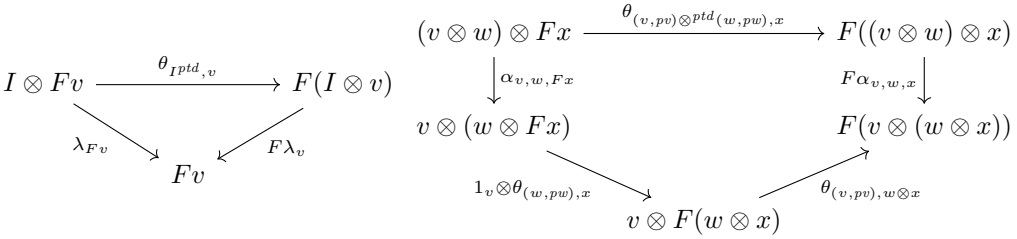


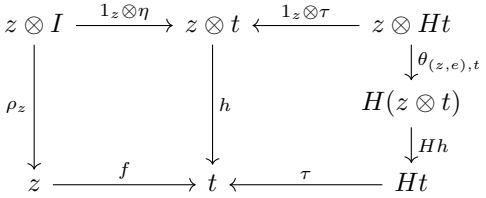
Figure 4 Preservation of the unitor and the actor for pointed tensorial strength.

In order to allow for an easy comparison with the literature, we spell out the lineator laws for pointed tensorial strength θ (besides the requirement of naturality in both arguments): the components are $\theta_{(v,pv),x} : v \otimes Fx \to F(v \otimes x)$, and the preservation rules are given in Figure 4. The differences with [19, Section 5.2.1] are all of presentational nature, most notably that we use left actegories while Hur has the monoidal-pointed objects as second parameter of his "pointed strength" st.

3 Monoid Structure on Non-Wellfounded Syntax

In this section, we construct a well-behaved substitution operation on non-wellfounded syntax. We do so on the level of monoidal categories, using the new notion of "monoidal heterogeneous substitution system". Already mentioned in the introduction, this notion will have a pivotal role in this section: as an intermediate step between a given final coalgebra and the monoid representing substitution on that final coalgebra. The carrier of these structures is one object t of the given monoidal category $\mathcal{V}$, so t is the representation of all terms as a whole, thus abstracting away from context/scope and typing details. (In Section 4, $\mathcal{V}$ will be instantiated to an endofunctor category, so that such a t will be a functor whose argument is interpreted as a typing context.)

▶ **Definition 1** (Monoidal heterogeneous substitution system, `mhss`). *Let $\mathcal{V}$ be a monoidal category with unit I, tensor $\otimes$ and right unitor ρ and H an endofunctor on $\mathcal{V}$ with a pointed tensorial strength θ for H. We consider triples (t, η, τ) with $t : \mathcal{V}$ (the "terms"), $\eta : I \to t$ (representing the injection of variables into terms) and $\tau : Ht \to t$ (the H-algebra representing the domain-specific constructors). Hence (t, η) is a monoidal-pointed object. (t, η, τ) is a* monoidal heterogeneous substitution system *(MHSS) for $(\mathcal{V}, H, \theta)$ if, for all (z, e, f) with $z : \mathcal{V}$, $e : I \to z$ and $f : z \to t$, there is a unique morphism $h : z \otimes t \to t$ such that the following diagram commutes:*



(1)

The uniquely existing morphism h is denoted as $(\!|f|\!)_{(z,e)}$.

Notice that for the considered triples (z, e, f), (z, e) is a monoidal-pointed object. The morphism f is just a $\mathcal{V}$-morphism and not a "monoidal-pointed" morphism from (z, e) to (t, η), see Remark 2. The left unitor and the associator of the monoidal category do not enter this definition directly but through the laws governing θ (cf. Figure 4).

As seen on the right-hand side of Equation (1), the strength θ is an operation that serves to prepare the arguments that are fed into the "structurally recursive call" Hh, before applying the domain-specific constructors bundled in τ. In other words, h mostly follows a homomorphic pattern, except for the rearrangement required by variable binding – implicitly expressed in functor H – that is taken care of by θ.

This notion of MHSS is not a recursion scheme specifically for the carrier of an initial algebra (for the functor $I + H-$). The present notion of MHSS only formulates the "desideratum", not sufficient conditions for its fulfillment. What makes MHSS suitable for dealing with coinductive syntax as well (when t, η, τ come from a final coalgebra) is the deliberate restriction of the target type of h to t. (This is already part of the notion of heterogeneous substitution system (HSS) [22], see Remark 2). There is also the restriction to τ as the H-algebra in the arrow on the bottom of the diagram, hence the limitation to a notion of substitution and not some general recursive pattern.

▶ Remark 2. *Monoidal* heterogeneous substitution systems "almost" generalize the notion of HSS of [22] from the specific situation where an endofunctor category $[\mathcal{C}, \mathcal{C}]$ is considered to the (unrestricted) monoidal category $\mathcal{V}$. The ingredients and stipulations of [22, Definition 5] are an instance of our notion of MHSS as soon as $\mathcal{C}$ has binary coproducts – so as to be able to speak about an $(\mathsf{Id} + H-)$-algebra – modulo the following:

1. The order of the arguments of strength θ is inverted.
2. We consider all $\mathcal{V}$-morphisms f and not only morphisms f between the monoidal-pointed objects (z, e) and (t, η), satisfying $\eta = e \cdot f$. In our diagram, that is Equation (1), f is just the $\mathcal{V}$-morphism. However, in the diagram in [22, Definition 5], f is written although Uf is meant, with U the forgetful functor from pointed endofunctors to endofunctors forgetting the points.[1]

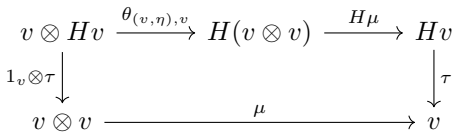
For the representation of substitution for non-wellfounded syntax, we will need abstract counterparts to the construction of a monad out of a HSS [22, Theorem 10] and the construction of a HSS from a final coalgebra [22, Theorem 17]. It is fair to say that these results carry over to MHSS without difficulty. We sketch the counterpart to the former construction in Section 3.1 and detail a different path to obtaining the latter in Section 3.2.

3.1 Construction of a Monoid From a MHSS

Let $(\mathcal{V}, \otimes, I, \lambda, \rho, \alpha)$ be a monoidal category. A $\mathcal{V}$-monoid is given by a triple (v, η, μ) where $v : \mathcal{V}$, $\eta : I \to v$ (the "unit" of the monoid) and $\mu : v \otimes v \to v$ (monoid "multiplication"), such that the left and right unit laws and the associative law hold. We recall the laws in Appendix B (only in [23]).

Let furthermore H be an endofunctor on $\mathcal{V}$ with a pointed tensorial strength θ for H. An (H, θ)-monoid [17] is a quadruple (v, η, μ, τ) with (v, η, μ) a $\mathcal{V}$-monoid and $\tau : Hv \to v$ (thus (v, τ) is an H-algebra), such that the following diagram commutes:

[1] This second difference appears to be a *conceptual* simplification, and has been formalized in 2022 for endofunctor categories as a "simplified notion of HSS", serving as a test bed for our definition of MHSS, cf. `SubstitutionSystems.SimplifiedHSS.SubstitutionSystems.v`.



(2)

The condition expressed by the diagram is the starting point for the parameterization process that ends in the definition of MHSS.

▶ **Theorem 3** (Construction of a monoid from MHSS, `mhss_monoid`, `mhss_to_sigma_monoid`). *We assume the parameters $\mathcal{V}$, H and θ of a MHSS. Let (t, η, τ) be a MHSS, and let $\mu := (\!|1_t|\!)_{(t,\eta)}$ the uniquely existing morphism for $(t, \eta, 1_t)$. Then (t, η, μ, τ) is an (H, θ)-monoid.*

This generalizes [17, Proposition 3.5] from the construction of an initial (H, θ)-monoid (under extra sufficient conditions) to the construction of an (H, θ)-monoid from a suitable H-algebra (without further conditions). On the other hand, it lifts [22, Theorem 10] from the abstraction level of endofunctor categories to that of monoidal categories. For the proof of Theorem 3, we can precisely follow the organization of the proof of [22, Theorem 10]. The absence of the pointedness requirement for f in the definition of MHSS gives rise to an inessential simplification. The defining diagram of an (H, θ)-monoid is just the right-hand side of Equation (1) for the instance used to define μ. So, we have to establish the monoid laws, for which we only give an overview (cf. `mhss_monoid` for the formalization). We define $\mu^{(0)} :\equiv \eta : I \to t$ and $\mu^{(1)} :\equiv (\!|\eta|\!)_{I^{ptd}} : I \otimes t \to t$. The morphism λ_t satisfies its defining diagram, hence $\mu^{(1)} = \lambda_t$ by uniqueness. The right unit law of a monoid is just the left-hand side of the defining diagram of μ. The left unit law of a monoid asks for λ_v to be equal to a morphism m; since $\lambda_t = \mu^{(1)}$, it suffices to show that m satisfies the defining diagram of $\mu^{(1)}$. Now, define $\mu^{(2)} :\equiv \mu$. The morphism $\mu^{(2)}$ is even a morphism of the monoidal-pointed objects $(t, \eta) \otimes^{ptd} (t, \eta)$ and (t, η); the proof uses that $\lambda_I = \rho_I$, which holds generally. Define $\mu^{(3)} :\equiv (\!|\mu^{(2)}|\!)_{(t,\eta)\otimes^{ptd}(t,\eta)} : (t \otimes t) \otimes t \to t$. The associative law of a monoid can now be dealt with by showing that both sides of that equation satisfy the defining diagram of $\mu^{(3)}$ and are hence equal by uniqueness. The reasoning in both cases is just the monoidal generalization of the first two items of [22, p. 168].

3.2 Construction of a MHSS From a Final Coalgebra

In this section, we assume the parameters $\mathcal{V}$, H and θ of a MHSS. We require binary coproducts in the underlying category of $\mathcal{V}$ (and use inl and inr without indices for the left and right injection into the coproduct). We also assume a final coalgebra (t, out) of the functor $(I + H-)$, i. e., $t : \mathcal{V}$ and $\mathsf{out} : t \to I + Ht$. By Lambek's theorem, out is an isomorphism, with inverse $\mathsf{out}^{-1} : I + Ht \to t$ that can be written as $\mathsf{out}^{-1} = [\eta, \tau]$ with $\eta : I \to t$ and $\tau : Ht \to t$. We also require that binary coproducts distribute over the tensor of $\mathcal{V}$ in its second argument; this means that, for all v, w_1 and w_2, the morphism $[1_v \otimes \mathsf{inl}, 1_v \otimes \mathsf{inr}]$ from $v \otimes w_1 + v \otimes w_2$ to $v \otimes (w_1 + w_2)$ has an inverse. We call that inverse δ for "distributor", without specifying its arguments.

▶ **Theorem 4** (Construction of MHSS from final coalgebra, `final_coalg_to_mhss_alt`). *The triple (t, η, τ) is a MHSS for $(\mathcal{V}, H, \theta)$.*

For clarity, we deviate from the proof of [22, Theorem 17] for HSS, which uses primitive corecursion. We instead use that out^{-1} is a completely iterative algebra (abbreviated as "cia"), which follows from out being a final coalgebra [25]. In particular, we will only use the definition of cia and not more general corecursion schemes implied by that property.

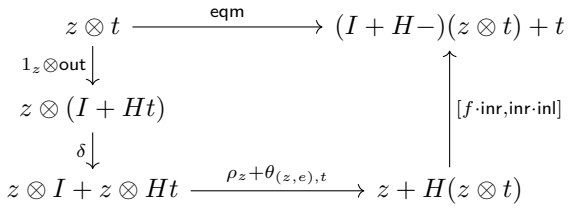
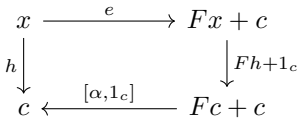


Figure 5 Definition of eqm as composition of four morphisms.

Given an endofunctor F on a category $\mathcal{C}$ with binary coproducts, an F-algebra (c, α) is called a cia iff for every $x : \mathcal{C}$ and every morphism $e : x \to Fx + c$ ("a flat equation morphism"), there is a unique morphism $h : x \to c$ that is a "solution" of e in c in the sense that the following diagramm commutes:



This generalizes the intuition when $\mathcal{C}$ is **Set**: the elements of x are the unknowns, and e either requires a structure in F over the unknowns or directly assigns a value in c. A morphism h is a solution if, in the first case, applying h inside the structure and then assembling the structure through α yields the value of h again.

To prove Theorem 4, we apply the cia scheme for $F := I + H-$ and $\alpha := \mathsf{out}^{-1}$. Given a triple (z, e, f), the following are equivalent:

1. $h : z \otimes t \to t$ satisfies the defining diagram for $(\!|f|\!)_{(z,e)}$
2. h satisfies the defining diagram of a solution for the flat equation morphism $\mathsf{eqm} : z \otimes t \to (I + H-)(z \otimes t) + t$ defined in Figure 5.

The details are found in Appendix C. In a nutshell, the two defining diagrams can be massaged so that the equivalence can be seen for each path in the diagrams individually.

On a general note, there is a whole arsenal of categorical corecursion schemes. For MHSS (and hence for the representation of substitution in the section to come), we picked the method of completely iterative algebras. Working with these tools from category theory is an alternative to intuitive "guarded" definitions and reasoning with observation depths. This alternative is suitable for formalization, for which the present paper is further evidence.

4 Non-Wellfounded Syntax for Multi-Sorted Binding Signatures

In this section, we start from the notion of multi-sorted binding signature (reviewed in Section 4.1). Exploiting the high-level results of Section 3 and thus showing their usefulness, we are going to construct the non-wellfounded syntax specified by such a signature, together with a well-behaved – monadic – substitution operation on the terms of that syntax.

Our work builds upon previous work [5] involving two of the present authors. There, categorical semantics of languages of *well*founded terms is developed, and a construction of the syntax generated by a multi-sorted binding signature is given. In this section, we construct *non*-wellfounded syntax based on that very same notion of multi-sorted binding signatures. Given such a signature, the existence of the generated syntax is guaranteed by ω-continuity of the associated signature functor – while, for wellfounded terms, [5] establishes ω-cocontinuity to construct the syntax. For a modular proof of ω-continuity, we decompose

the construction of the associated signature functor slightly differently. Extensionally, we arrive at the same functor, but the formalization of that proof is somewhat intricate. We therefore suggest the new construction of the signature functor as the one to work with also in the wellfounded case. This makes good sense if one wants to consider the embedding of wellfounded syntax into non-wellfounded syntax for the same multi-sorted binding signature, and it is doable since we also formalized a proof of ω-cocontinuity. But there is also an advantage on the conceptual side: the building blocks of the signature functor are all endofunctors, unlike previously [5]. A second difference with the previous work is that, for the strength construction, we systematically refer to results that reside on the abstract level of monoidal categories. (Aspects of steps 1, 2, and 4 below are also described in [5, Section 2]; we discuss them here again for the sake of being self-contained. The items 5-7 are concretized in Section 4.4.)

1. We describe simply-typed syntax with variable binding (of finitely many sorted variables in each constructor argument) as a multi-sorted binding signature, see Section 4.1.
2. Given a multi-sorted binding signature, we construct a signature functor H (deviating from [5] for technical reasons), see Section 4.2.
3. We prove ω-continuity of $(\mathsf{Id} + H-)$ and construct the coinductive syntax as the inverse of a final coalgebra thereof, see Section 4.3.
4. We construct a "lax lineator" between actions expressing pointed tensorial strength of H, see Section 4.4 including a discussion of the case of simply-typed λ-calculus and references to the appendix for more details on the general case (only in [23]).
5. We construct a MHSS (Definition 1) for (H, θ) by applying Theorem 4.
6. We construct an (H, θ)-monoid by applying Theorem 3.
7. Finally we interpret the obtained monoid as monad (hence as monadic substitution) since, during the entire section, we are instantiating the monoidal category to the endofunctors.

4.1 Multi-Sorted Binding Signatures: Motivation and Definition

We want to construct syntax of non-wellfounded terms that feature variable binding and have a simple notion of typing. Such type systems can be specified using "multi-sorted binding signatures"; this notion was used, in particular, in [5], but appears in almost any literature about initial semantics for multi-sorted syntax. The prime example is simply-typed λ-calculus (STLC), whose extension to non-wellfounded well-typed terms is an instance of our construction. We study this example in some detail in Example 5 before reviewing multi-sorted binding signatures in Definition 6.

▶ **Example 5** (Non-wellfounded simply-typed lambda-calculus)**.** We are now rephrasing [5, Example 2.2 and Example 2.10]. We assume the types of simply-typed λ-calculus to form a small set S that is closed under a binary operation $\Rightarrow\colon \mathsf{S} \to \mathsf{S} \to \mathsf{S}$. The elements of S are called *sorts*, so as to distinguish them from the types of our ambient type theory. We model syntax over a base category $\mathcal{C}$ (with initial object $\bot$, terminal object $\top$, and binary products and binary coproducts), not necessarily the category Set; however, we motivate the notions for the special case where $\mathcal{C}$ is Set. Let $\mathcal{C}^{\mathsf{S}}$ be the functor category $[\mathsf{S}, \mathcal{C}]$ where S is viewed as a discrete category. In the case when $\mathcal{C}$ is Set, objects of this category are simply functions $\xi : \mathsf{S} \to \mathsf{Set}$, and we generally use letter ξ to indicate objects of $\mathcal{C}^{\mathsf{S}}$. They represent the typing contexts since ξs represents the totality of variables of sort s.

For the instantiation of Section 3, we take $\mathcal{V}$ as the monoidal category of endofunctors of $\mathcal{C}^{\mathsf{S}}$ – with the tensor operation $X \otimes Y :\equiv X \cdot Y$ in diagrammatic order. In Definition 1, we are looking for one object T of $\mathcal{V}$ (i. e., $T : [\mathcal{C}^{\mathsf{S}}, \mathcal{C}^{\mathsf{S}}]$) as representation of all the wellfounded and non-wellfounded terms, here of simply-typed λ-calculus. On objects, T assigns to $\xi : \mathcal{C}^{\mathsf{S}}$

and $s : \mathsf{S}$ the object $T\ \xi\ s$ of $\mathcal{C}$, a representation of all the wellfounded and non-wellfounded terms that have sort s in the typing context ξ. The functor H of Section 3 prepares for the construction of T as a fixed point. Instead of only considering the "solution" T as an argument to H, we have to abstract over an arbitrary $X : \mathcal{V}$ as an object argument to H. We would like to take H as the pointwise coproduct of one summand for application and one for abstraction, for each pair (s, t) of sorts that parameterize the respective typing rules, i. e. $H :\equiv \sum_{s,t:\mathsf{S}} (\mathsf{app}_{s,t} + \mathsf{lam}_{s,t})$. Here the summands have to be endofunctors on $[\mathcal{C}^{\mathsf{S}}, \mathcal{C}^{\mathsf{S}}]$, and we only give the definition for objects (in all arguments) – where the defining equation is between objects of $\mathcal{C}$:

$$\mathsf{app}_{s,t}\ X\ \xi\ u :\equiv \begin{cases} X\ \xi\ (s \Rightarrow t) \times X\ \xi\ s & \text{if } u = t \\ \bot & \text{else} \end{cases},$$

$$\mathsf{lam}_{s,t}\ X\ \xi\ u :\equiv \begin{cases} X\ \xi'\ t & \text{if } u = (s \Rightarrow t), \text{ with } \xi' s :\equiv \top + \xi s \text{ and } \xi' u' = \xi u' \text{ for } u' \neq s \\ \bot & \text{else} \end{cases}$$

These summands represent all well-sorted applications and λ-abstractions made from the material in the yet arbitrary object X of $\mathcal{V}$, while the variables are dealt with separately from H through the unit of $\mathcal{V}$ (in our case the identity functor) in Section 3. The case distinctions on equality of sorts in this motivating example can be avoided following [5], and we will do so in the generic construction in Section 4.2.

We fix a small set S representing the sorts.

▶ **Definition 6** ([5, Definition 2.1], `MultiSortedSig`)**.** *A* multi-sorted binding signature *is given by a small set I together with an arity function* $\mathsf{ar} : I \to (\mathsf{S}^* \times \mathsf{S})^* \times \mathsf{S}$.

Here, we write A^* for the set of finite lists formed from elements of A. The intuition is as follows: for any $i : I$, $\mathsf{ar}(i)$ is the signature of a term constructor. The second component of $\mathsf{ar}(i)$ is the sort of the constructed term. The first component is a list of signatures of the arguments of that constructor. Each such signature is an element of $\mathsf{S}^* \times \mathsf{S}$, describing the sorts of all the (anonymous) variables bound by that argument, together with the sort of the argument itself. [5] makes no claim on originality of that definition, see the discussion there. It should be stressed that, while S and I can be infinite sets, each term constructor described by an $\mathsf{ar}(i)$ only has finitely many arguments. Non-wellfounded syntax with these constructors is therefore still finitary in the sense that terms, when viewed as trees, are finitely branching.

▶ **Example 7** ([5, Example 2.2], `STLC_Sig`)**.** Assume that S is closed under a binary operation $\Rightarrow$. We put into I the sort parameters of the typing rules of the term constructors of STLC. Thus, I is taken to be $(\mathsf{S} \times \mathsf{S}) + (\mathsf{S} \times \mathsf{S})$. The left summand pertains to the application operation while the right summand describes λ-abstraction:

$$\mathsf{ar}(\mathsf{inl}\langle s, t\rangle) :\equiv \big\langle [\langle [], s \Rightarrow t\rangle, \langle [], s\rangle], t \big\rangle \qquad \mathsf{ar}(\mathsf{inr}\langle s, t\rangle) :\equiv \big\langle [\langle [s], t\rangle], s \Rightarrow t \big\rangle$$

▶ **Example 8** (`UntypedForest_Sig`)**.** We model the grammar of the untyped version of the forests described in Section 1.2 as a multi-sorted binding signature. Let $\mathsf{S}_0 := \{\mathsf{v}, \mathsf{t}, \mathsf{e}\}$ be a three-element set, having sorts for the three syntactic categories of term variables (v), terms (t) and elimination alternatives (e). The first sort seems unavoidable since the elimination alternative $x\langle N_1, \ldots, N_k\rangle$ only allows term variables in the head position, not arbitrary terms. The index set I represents the different forms of expressions that are parameterized in the elements of the syntactic categories: one for λ-abstraction, one for each index n for

summation, one for each index k for tupling, hence we set $I := (1 + \mathbb{N}) + \mathbb{N}$. We define $\mathsf{ar}(\mathsf{inl}(\mathsf{inl}*)) :\equiv \langle[\langle[\mathsf{v}], \mathsf{t}\rangle], \mathsf{t}\rangle$, a simplified version of the second case for STLC: a term variable is being bound in a term, yielding a term. The other forms of expressions do not feature variable binding:

$$\mathsf{ar}(\mathsf{inl}(\mathsf{inr}\, n)) :\equiv \langle[\underbrace{\langle[], \mathsf{e}\rangle, \ldots, \langle[], \mathsf{e}\rangle}_{n}], \mathsf{t}\rangle \qquad \mathsf{ar}(\mathsf{inr}\, k) :\equiv \langle[\langle[], \mathsf{v}\rangle, \underbrace{\langle[], \mathsf{t}\rangle, \ldots, \langle[], \mathsf{t}\rangle}_{k}], \mathsf{e}\rangle$$

None of the arities have v as second component, hence term variables will only come from a given context. Using the pipeline of Section 4.4, we can represent these untyped forests as an object T of $[\mathcal{C}^{\mathsf{S}_0}, \mathcal{C}^{\mathsf{S}_0}]$, analogously to Example 5. Again analogously to that example, objects ξ of $\mathcal{C}^{\mathsf{S}_0}$ represent contexts; in this untyped version they correspond just to a choice of names for all occurring variables. Then, because of the absence of v as second component of our arities, we should have that $T\xi\mathsf{v}$ and $\xi\mathsf{v}$ are isomorphic for any $\xi : \mathcal{C}^{\mathsf{S}_0}$. In our representation, we are interested in those $\xi : \mathcal{C}^{\mathsf{S}_0}$ for which $\xi\mathsf{t}$ and $\xi\mathsf{e}$ are the initial object of $\mathcal{C}$ – empty sets in the case of $\mathcal{C} = \mathbf{Set}$ – so that ξ only provides names for term variables. For those ξ, the untyped forests in the terms category and elimination alternatives category, of the grammar in Section 1.2, are represented as $T\xi\mathsf{t}$ and $T\xi\mathsf{e}$, respectively.

▶ **Example 9** (`Forest_Sig`)**.** The typed forests of Section 1.2 have as the set of sorts the set $\mathsf{S} \times \mathsf{S}_0$, with S and S_0 from the two previous examples. Moreover, since atomic types play a specific role in the typing rules in Figure 1, we have to assume a set atom of atoms and an operation $\mathsf{atotype} : \mathsf{atom} \to \mathsf{S}$ (which should be thought of as an inclusion). The typing rule for tuples is additionally (as compared to the raw syntax) parameterized by a list of k elements of S and one element of atom. Accordingly, the index set for this multi-sorted binding signature is $I := (\mathsf{S} \times \mathsf{S} + \mathsf{atom} \times \mathbb{N}) + \mathsf{S}^* \times \mathsf{atom}$, using the set S^* of finite S-lists introduced above. We define $\mathsf{ar}(\mathsf{inl}(\mathsf{inl}\langle s, t\rangle)) :\equiv \langle[\langle[\langle s, \mathsf{v}\rangle], \langle t, \mathsf{t}\rangle\rangle], \langle s \Rightarrow t, \mathsf{t}\rangle\rangle$, which combines the second case of Example 7 and the first case of Example 8. The other definitions are as follows:

$$\mathsf{ar}(\mathsf{inl}(\mathsf{inr}\, \langle p, n\rangle)) :\equiv \langle[\underbrace{\langle[], \langle \mathsf{atotype}\, p, \mathsf{e}\rangle\rangle, \ldots, \langle[], \langle \mathsf{atotype}\, p, \mathsf{e}\rangle\rangle}_{n}], \langle \mathsf{atotype}\, p, \mathsf{t}\rangle\rangle$$

$$\mathsf{ar}(\mathsf{inr}\, \langle[B_1, \ldots, B_k], p\rangle) :\equiv \langle[\langle[], \langle B, \mathsf{v}\rangle\rangle, \langle[], \langle B_1, \mathsf{t}\rangle\rangle, \ldots, \langle[], \langle B_k, \mathsf{t}\rangle\rangle], \langle \mathsf{atotype}\, p, \mathsf{e}\rangle\rangle \ ,$$

with $B := B_1 \Rightarrow \ldots \Rightarrow B_k \Rightarrow \mathsf{atotype}\, p$, parenthesized to the right.

A multi-sorted binding signature is just simple syntactic data (of a signature), so rather the description of a task to define the intended syntax – which in our case will include non-wellfounded terms. In the next section we discuss how to transform such a signature into a more "semantic" kind of signature: a functor H such that the semantics of the signature is given by $(\mathsf{Id} + H-)$-algebras. While we did this "by hand" for STLC in Example 5 (but limited the description just to object arguments), it should be clear that an automatic generation for more involved grammars such as Example 9 would be desirable.

4.2 (Modified) Signature Functor for Multi-Sorted Binding Signatures

Here, we associate to any multi-sorted binding signature (I, ar) a suitable functor H, such that the non-wellfounded syntax generated by (I, ar) is, in particular, an $(\mathsf{Id} + H-)$-(co)algebra. Definitions 10–13 provide the building blocks for building H modularly from basic constructions, following the combinatorial structure of (I, ar) as a family of pairs containing lists.

We now use $\mathcal{C}^{\mathsf{S}}$ generally for the functor category $[\mathsf{S}, \mathcal{C}]$, not just for the STLC example in Section 4.1. One can still think of $\mathcal{C}$ as being Set, but we keep the category $\mathcal{C}$ abstract and collect requirements on $\mathcal{C}$ on the way – that are all fulfilled by Set. As mentioned in Section 4.1, we instantiate $\mathcal{V}$ of Section 3 with the endofunctors on $\mathcal{C}^{\mathsf{S}}$ and now have to determine the endofunctor H on $[\mathcal{C}^{\mathsf{S}}, \mathcal{C}^{\mathsf{S}}]$.

In order to do this, we assume that $\mathcal{C}$ has a terminal object $\top$, binary products, and set-indexed coproducts (including initial object $\bot$ and binary coproducts).

▶ **Definition 10** ([5, Definition 2.3], `sorted_option_functor`). *Let s be a sort. The* sorted option functor $\mathsf{option}_s : \mathcal{C}^{\mathsf{S}} \to \mathcal{C}^{\mathsf{S}}$ *is defined (on objects) as* $\mathsf{option}_s\ \xi\ t :\equiv \coprod_{(s=t)} \top + \xi\ t$.

In this definition, we form a coproduct in $\mathcal{C}$ of $\top$ over the type of proofs that $s = t$; i. e., we form a subsingleton. We thus avoid the use of case distinction: $\mathsf{option}_s\ \xi$ is an equivalent replacement for ξ' in the definition of lam above [5, Remark 2.4, Remark 2.8].

▶ **Definition 11** ([5, Definition 2.5], `option_list`). *Given a non-empty list of sorts $\ell \equiv [s_1, \ldots, s_n]$,* $\mathsf{option}^*\ \ell : \mathcal{C}^{\mathsf{S}} \to \mathcal{C}^{\mathsf{S}}$ *is defined as* $\mathsf{option}^*\ \ell :\equiv \mathsf{option}_{s_1} \cdot (\mathsf{option}_{s_2} \cdot \ldots)$. *For an empty list, it is* $\mathsf{option}^*\ [] :\equiv \mathsf{Id}$.

▶ **Definition 12** ([5, Definition 2.6], `projSortToC`). *For any $s : \mathsf{S}$ the* projection functor $\mathsf{pr}_s : \mathcal{C}^{\mathsf{S}} \to \mathcal{C}$ *is defined (on objects) as:* $\mathsf{pr}_s\ \xi :\equiv \xi\ s$.

▶ **Definition 13** ([5, Definition 2.7], `hat_functor`). *For any $s : \mathsf{S}$ we have a left adjoint to pr_s, written $\hat{s} : \mathcal{C} \to \mathcal{C}^{\mathsf{S}}$, defined on objects as* $\hat{s}\ c\ t :\equiv \coprod_{(s=t)} c$.

Here, we essentially define $\hat{s}\ c\ s = c$ and $\hat{s}\ c\ t = \bot$ otherwise. As above, we avoid the case analysis for $\mathsf{app}_{s,t}\ X\ \xi\ u$ in our STLC example in Section 4.1, hence do not need the matching of the constructor's target type.

We now have all the basic building blocks to associate, to a given multi-sorted binding signature (I, ar), a signature functor $H : [\mathcal{C}^{\mathsf{S}}, \mathcal{C}^{\mathsf{S}}] \to [\mathcal{C}^{\mathsf{S}}, \mathcal{C}^{\mathsf{S}}]$. We turn to the construction of the corresponding building blocks for H, involving a formal argument $X : [\mathcal{C}^{\mathsf{S}}, \mathcal{C}^{\mathsf{S}}]$ to H. Intuitively, X is the unknown functor that, after applying our results of Section 3, will be set to the functor representing the coinductive sorted syntax. For the sake of motivating the modification of the definition of H compared to [5], we will now work top-down and use the letter G with upper indices (instead of F with upper indices, used there).

The final step of the construction of H is unchanged from [5]. Assume that we already have a functor $G^{(\vec{a},t)} : [\mathcal{C}^{\mathsf{S}}, \mathcal{C}^{\mathsf{S}}] \to [\mathcal{C}^{\mathsf{S}}, \mathcal{C}^{\mathsf{S}}]$ for all $(\vec{a}, t) : (\mathsf{S}^* \times \mathsf{S})^* \times \mathsf{S}$. Then, for the multi-sorted binding signature (I, ar), the associated signature functor $H : [\mathcal{C}^{\mathsf{S}}, \mathcal{C}^{\mathsf{S}}] \to [\mathcal{C}^{\mathsf{S}}, \mathcal{C}^{\mathsf{S}}]$ is given by the (pointwise) coproduct $HX :\equiv \coprod_{i\ :\ I} G^{\mathsf{ar}(i)} X$ (`MultiSortedSigToFunctor'`).

The penultimate step constructs $G^{(\vec{a},t)}$ (`hat_exp_functor_list'_optimized`). It assumes given functors $G^{(a,t)} : [\mathcal{C}^{\mathsf{S}}, \mathcal{C}^{\mathsf{S}}] \to [\mathcal{C}^{\mathsf{S}}, \mathcal{C}^{\mathsf{S}}]$ for all $(a, t) : (\mathsf{S}^* \times \mathsf{S}) \times \mathsf{S}$. Given a non-empty list $\vec{a} \equiv [a_1, \ldots, a_n]$, $G^{(\vec{a},t)}$ is defined (on objects) as the iterated pointwise binary product $G^{(a,t)}X :\equiv G^{(a_1,t)}X \times (G^{(a_2,t)}X \times \ldots)$. The corner case $G^{([],t)}$ is given (maybe peculiarly) by $G^{([],t)}X :\equiv \top_{[\mathcal{C}^{\mathsf{S}},\mathcal{C}]} \cdot \hat{t}$, regardless of X, with $\top_{\mathcal{D}}$ the terminal object of $\mathcal{D}$. (The terminal object in functor categories is given by constantly the terminal object of the target category.) More precisely, $G^{([],t)}$ is the composition (in diagrammatic order) of first $\top_{[[\mathcal{C}^{\mathsf{S}},\mathcal{C}^{\mathsf{S}}],[\mathcal{C}^{\mathsf{S}},\mathcal{C}]]}$ and second post-composition with $\hat{t}$ (which is a functor from $[\mathcal{C}^{\mathsf{S}}, \mathcal{C}]$ to $[\mathcal{C}^{\mathsf{S}}, \mathcal{C}^{\mathsf{S}}]$). This view will be exploited for the construction of a pointed tensorial strength for $G^{([],t)}$.

It remains to construct $G^{(a,t)}$ (`hat_exp_functor_list'_piece`). It is a refined version of F^a in [5] since it takes into account the "target" sort t. In fact $G^{(a,t)}$, corresponds to $F^a \cdot \hat{t}$ in terms of that paper, and so the difference between our $G^{(\vec{a},t)}$ and $F^{(\vec{a},t)}$ in that

paper is whether the composition with $\hat{t}$ is on each component of the product (our solution) or only on the product itself. That previous solution looks less convoluted but requires the consideration of non-endofunctors.

Let $a \equiv (\ell, s)$ with $\ell : \mathsf{S}^*$. The object part of functor $G^{(a,t)}$ is defined as $G^{(a,t)}X :\equiv \mathsf{option}^*\ \ell \cdot X \cdot \mathsf{pr}_s \cdot \hat{t}$. More precisely, $G^{(a,t)}$ is the composition (in diagrammatic order) of first the precomposition with $\mathsf{option}^*\ \ell$ and second the postcomposition with $\mathsf{pr}_s \cdot \hat{t}$. This view will be important to establish ω-continuity of $G^{(a,t)}$.

The instance for STLC can be compared to our introductory example in Section 4.1 – assuming decidability of the set of sorts, they coincide mathematically.

4.3 Existence of Final Coalgebra for Functor $\mathsf{Id} + H-$

Given a multi-sorted binding signature, we want to apply Theorem 4 with $\mathcal{V}$ the endofunctors on $\mathcal{C}^{\mathsf{S}}$ and the signature functor H defined in the previous section. Its first requirement is a final coalgebra of the functor $(\mathsf{Id} + H-)$, with Id the identity functor on $\mathcal{C}^{\mathsf{S}}$ (which is the unit of $\mathcal{V}$). We get a final coalgebra through the dual of Adámek's theorem on the existence of initial algebras for ω-cocontinuous functors on ω-cocomplete categories with initial object. We now require that $\mathcal{C}$ is ω-complete (i. e., $\mathcal{C}$ has limits of shape $0 \leftarrow 1 \leftarrow 2 \leftarrow \cdots$), whence $\mathcal{V}$ is also ω-complete. (We had already generally required a terminal object $\top$ for $\mathcal{C}$, which gives one for $\mathcal{V}$.) We argue first that H is ω-continuous, then that $(\mathsf{Id} + H-)$ is ω-continuous.

We analyze the building blocks of H. $G^{(a,t)}$ is defined as the composition of two functors, so it is ω-continuous if both functors are. We prove that postcomposition with $\mathsf{pr}_s \cdot \hat{t}$ is ω-continuous (`post_comp_with_pr_and_hat_is_omega_cont`). For this, we require of $\mathcal{C}$ that ω-limits distribute over sub-singleton coproducts. This means that the canonical morphism from the coproduct of the respective limits to the limit of the coproducts is an isomorphism. Precomposition with any fixed functor is ω-continuous, hence in particular for option^*_ℓ.

We move to ω-continuity of $G^{(\vec{a},t)}$, which we prove by induction on the length of $\vec{a}$. For an empty $\vec{a}$, this is a constant functor; and pointwise binary products preserve ω-continuity.

Moreover, we require of $\mathcal{C}$ that ω-limits distribute over I-coproducts, a property that then also holds of $\mathcal{V}$. Then, ω-continuity of the $G^{\mathsf{ar}(i)}$ carries over to their coproduct H.

As a final step, we need to show ω-continuity of $(\mathsf{Id} + H-)$ (`is_omega_cont_Id_H`). In order to avoid still other hypotheses about distribution of ω-limits over certain colimits, we use that binary coproducts are (isomorphic) to bool-indexed coproducts. So, the final assumption on $\mathcal{C}$ is that ω-limits distribute also over bool-coproducts (besides over sub-singleton coproducts and I-coproducts for the index set I of the multi-sorted binding signature).

For the most interesting case $\mathcal{C} = \mathbf{Set}$, all these requirements are met.

4.4 Putting Everything Together

We can now apply the general results of Section 3 for the construction of coinductive syntax with monadic substitution, as specified by a multi-sorted binding signature. We fix a set S of sorts and a multi-sorted binding signature (I, ar) over S. This means in particular that we take as $\mathcal{V}$ the endofunctors on $\mathcal{C}^{\mathsf{S}}$ when instantiating the results of Section 3. We denote by $H : [\mathcal{C}^{\mathsf{S}}, \mathcal{C}^{\mathsf{S}}] \to [\mathcal{C}^{\mathsf{S}}, \mathcal{C}^{\mathsf{S}}]$ the signature functor associated to (I, ar) according to the construction in Section 4.2.

We aim to apply Theorem 4 to the final $(\mathsf{Id} + H-)$-coalgebra obtained in Section 4.3. To this end, the parameter θ of a MHSS has to be specified, i. e., we have to construct a suitable pointed tensorial strength θ for the signature functor H. To recall, θ instructs via the notion

of (H, θ)-monoid (Equation (2)) how the monoid multiplication μ acts on the constructors (embodied in the H-algebra τ). But in our situation, μ is the monad multiplication expressing substitution – it is well-known that $\mathcal{V}$-monoids in this case are nothing but monads on $\mathcal{C}^{\mathsf{S}}$ (cf. `monoid_to_monad_CAT`). In other words, we have to define θ so that it describes correctly the recursive behaviour of substitution (as expressed by μ).

▶ **Example 14** (Strength for STLC)**.** A suitable θ for the "hand-written" signature functor H in Example 5 contains, in particular, operations from $Y \cdot (\mathsf{app}_{s,t} X)$ to $\mathsf{app}_{s,t}(Y \cdot X)$ and from $Y \cdot (\mathsf{lam}_{s,t} X)$ to $\mathsf{lam}_{s,t}(Y \cdot X)$ for any $s, t : \mathsf{S}$, endofunctor Y on $\mathcal{C}^{\mathsf{S}}$ that has a point $\eta : \mathsf{Id} \to Y$, and endofunctor X on $\mathcal{C}^{\mathsf{S}}$. The instance relevant in Equation (2) is then with X and Y the representation T of STLC, and with η the inclusion of variables in terms T. The operations for *application* do essentially nothing since μ should just descend into the subtrees. For *abstraction*, given a $\xi : \mathcal{C}^{\mathsf{S}}$, we need to specify a function of type $T\xi + 1_s \to T(\xi + 1_s)$, where $1_s : \mathcal{C}^{\mathsf{S}}$ is defined as $1_s(t) \coloneqq (t = s)$; that is, $X + 1_s$ is X extended by an element of sort s. In other words, we need to lift $T\xi : \mathcal{C}^{\mathsf{S}}$ extended by an element $\star$ of sort s to $T\xi'$, with ξ' the extension of ξ by an element $\star$ of sort s (as in Example 5); this will need η for $\star$ in the input. In short, for abstraction, the strength specifies essentially the famous "lift" operation on a substitution function to avoid capture when descending under a binder.

The construction of a suitable θ for the variant of the generic H considered in [5] could have been adapted to our H, but we have preferred to give a construction that, although working on the level of endofunctors, is formed from building blocks that reside on the general level of monoidal categories. To structure this construction, we propose the notion of *relative lax commutator* that generalizes the notion of "pointed distributive law" in [4, Definition 10]. For lack of space, the whole (technical) construction is explained in Appendix D, but only in [23]. We thus take as θ the pointed tensorial strength for H described in the appendix. For STLC (Example 7), the strength can be exploited on the abstract level with base category $\mathcal{C}$ (cf. `thetaSTLC` in the formalization). For forests (Example 8 and Example 9), we only exploited the situation with $\mathcal{C}$ set to **Set** (cf. `thetaUntypedForest` and `thetaForest` in the formalization).

Theorem 4 provides us with a MHSS that serves as input to Theorem 3, hence we get an (H, θ)-monoid. This monoid is, in particular, a monad, our (certified) substitution monad for the non-wellfounded syntax described by the given multi-sorted binding signature. For STLC, we have formalized this on the abstract level and constructed the finite Church numerals as well as the infinite Church numeral (cf. `SubstitutionSystems.STLC_actegorical_abstractcat.v`).

To be more concrete, we can instantiate the base category $\mathcal{C}$ to **Set** that satisfies all requirements on $\mathcal{C}$ we made during the construction process and get a set of well-sorted non-wellfounded terms for any sort, given a supply of sets of variables for any sort, together with a substitution operation that respects sorts and satisfies the monad laws. We replayed the construction of the finite Church numerals in STLC in this concrete setting (cf. `SubstitutionSystems.STLC_actegorical.v`). For forests, we only considered the **Set** case and have instantiated the general constructions (cf. `SubstitutionSystems.UntypedForests.v` and `SubstitutionSystems.Forests.v`).

Although this not the topic of this paper, we mention that we also have adapted the results and the formalization of wellfounded syntax to the present setting: This includes the construction of a MHSS from an initial $(I + H-)$-algebra under the proviso it has been obtained through a Mendler-style construction based on ω-cocontinuity of H (`initial_alg_to_mhss`). We further established that this MHSS gives rise to an initial (H, θ)-monoid `SigmaMonoidFromInitialAlgebraInitial`, for the given strength θ. These results are on

the level of a monoidal category, as in our Section 3. The signature functor H constructed in Section 4.2 is even ω-cocontinuous (based on the proof of the same property for the variant considered in [5]), under conditions on $\mathcal{C}$ that are fulfilled for **Set**. Thus, $(\mathsf{Id} + H -)$ is ω-bicontinuous, and we get a morphism of (H, θ)-monoids from the inductive to the coinductive syntax (`ind_into_coind` for the case **Set**). The above-mentioned four `Coq` vernacular example files in our `UniMath` library illustrate that, thanks to that actegorical development, the use of the formalized wellfounded and the formalized non-wellfounded syntax for those multi-sorted binding signatures can be done in parallel. For example, the finite Church numerals in STLC are developed independently of the choice for one of these two options. This conforms to the intuition that every single wellfounded term belonging to the non-wellfounded syntax already belongs to the wellfounded syntax, even though the categorical development of these structures is very different.

5 Related Work and Conclusions

We have cited throughout the paper the work we rely on or which initiated a line of thought. Here, we give additional information on other related work (that may have been also cited already in the main text). [20] also have codatatypes and define datatype-generic substitution corecursively, and they even calculate infinitary normal forms for their example of untyped λ-calculus. However, they do not consider typed systems, and the results are not presented on the abstraction level of monoidal categories. Instead, they use a concrete "nominal" presentation of syntax with binders. [6] also have codatatypes and even datatype-generic programming not only of substitution, but the work is not based on category theory (and so the approach is rather axiomatic than definitional). That work is implemented in the Agda system. [30] considers different categorical models of simply-typed wellfounded syntax. In its Chapter 5, the monoidal category corresponding to the framework of [17] is laid out in detail for simple types, and its Chapter 7 compares it with the monoidal category of endofunctors over a slice category. The latter is close to the concrete instance we are studying in Section 4, but we deal with non-wellfounded syntax. All in all, [30] has a lot on the strength construction with actegories, including for the typed case, and this for more than one concrete categorical representation, but non-wellfounded syntax is not considered. [12] have an approach to codatatypes that is definitionally based on category theory; but it is strongly tied to set theory through infinite cardinal numbers that appear in the definition of the class of "bounded natural functors" they consider. This allows them to implement the approach in the Isabelle system (based on a very small kernel). Popescu [26] compares different corecursors for syntax with variable binding in nominal style; it is partially formalized in Isabelle/HOL. [15] also translate multi-sorted binding signatures into signatures with strength. Their notion of syntax includes "meta-variables", but they stay within the wellfounded terms and heavily use inductive families as provided by the Agda system. [13] comes with a `UniMath/Coq` formalization of the whole chain even from *skew* monoidal categories to an initial (H, θ)-monoid, hence for wellfounded syntax. Beware that swapping of the arguments of the tensor is not a harmless operation for skew monoidal categories, so our present definition of MHSS does not fit as a pivotal element in their development. [18] rework the approach of [17], still using pointed strength and (H, θ)-monoids. They also deal with simple types but do not consider non-wellfounded syntax.

We have presented, through the notion of monoidal heterogeneous substitution system, a tool which provides a monadic substitution operation also for non-wellfounded syntax, and this for the first time on the abstraction level of monoidal categories.

Our definitions and results unify the construction of both wellfounded and non-wellfounded syntax with substitution.

We also instantiated monoidal heterogeneous substitution systems to endofunctor categories and adapted the full chain from multi-sorted binding signatures to substitution for non-wellfounded syntax. For the sake of this instantiation, we provide modular results to prove ω-cocontinuity of signature functors and hence obtain both ω-cocontinuity and ω-continuity for the signature functors we generate from multi-sorted binding signatures.

All the results of this paper have been rigorously formalized with `UniMath/Coq`. For the specific category of sets (types of homotopy level 2 according to univalent foundations) as base category, the hypotheses of the construction of non-wellfounded syntax can be proved. Hence, for this base category, we have a "concrete" formalization of the tool chain, which provides in particular a formal construction in univalent foundations of non-wellfounded syntax with binding, as instructed by a multi-sorted binding signature, and its monadic substitution operation.

A question we have left open is that of equations on non-wellfounded terms, for instance, β-equivalence. We anticipate that some definitions could carry over from the wellfounded setting, like the definition of equations and reductions given in [2, 3]. The construction of suitable terminal coalgebras, however, seems to require some work.

References

1 Benedikt Ahrens, Paolo Capriotti, and Régis Spadotti. Non-wellfounded trees in homotopy type theory. In Thorsten Altenkirch, editor, *13th International Conference on Typed Lambda Calculi and Applications, TLCA 2015, July 1-3, 2015, Warsaw, Poland*, volume 38 of *LIPIcs*, pages 17–30. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.TLCA.2015.17.

2 Benedikt Ahrens, André Hirschowitz, Ambroise Lafont, and Marco Maggesi. Modular specification of monads through higher-order presentations. In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany*, volume 131 of *LIPIcs*, pages 6:1–6:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.FSCD.2019.6.

3 Benedikt Ahrens, André Hirschowitz, Ambroise Lafont, and Marco Maggesi. Reduction monads and their signatures. *Proc. ACM Program. Lang.*, 4(POPL):31:1–31:29, 2020. doi:10.1145/3371099.

4 Benedikt Ahrens, Ralph Matthes, and Anders Mörtberg. From signatures to monads in UniMath. *J. Autom. Reason.*, 63(2):285–318, 2019. doi:10.1007/s10817-018-9474-4.

5 Benedikt Ahrens, Ralph Matthes, and Anders Mörtberg. Implementing a category-theoretic framework for typed abstract syntax. In Andrei Popescu and Steve Zdancewic, editors, *CPP '22: 11th ACM SIGPLAN International Conference on Certified Programs and Proofs, Philadelphia, PA, USA, January 17 - 18, 2022*, pages 307–323. ACM, 2022. doi:10.1145/3497775.3503678.

6 Guillaume Allais, Robert Atkey, James Chapman, Conor McBride, and James McKinna. A type- and scope-safe universe of syntaxes with binding: their semantics and proofs. *J. Funct. Program.*, 31:e22, 2021. doi:10.1017/S0956796820000076.

7 Thorsten Altenkirch and Bernhard Reus. Monadic presentations of lambda terms using generalized inductive types. In Jörg Flum and Mario Rodríguez-Artalejo, editors, *Computer Science Logic, 13th International Workshop, CSL '99, 8th Annual Conference of the EACSL, Madrid, Spain, September 20-25, 1999, Proceedings*, volume 1683 of *Lecture Notes in Computer Science*, pages 453–468. Springer, 1999. doi:10.1007/3-540-48168-0_32.

8 Hendrik Pieter Barendregt, Wil Dekkers, and Richard Statman. *Lambda Calculus with Types*. Perspectives in logic. Cambridge University Press, 2013. URL: http://www.cambridge.org/de/academic/subjects/mathematics/logic-categories-and-sets/lambda-calculus-types.

9 Henning Basold. *Mixed Inductive-Coinductive Reasoning—Types, Programs and Logic*. PhD thesis, Radboud University, Nijmegen, The Netherlands, 2018. URL: `https://repository.ubn.ru.nl/handle/2066/190323`.

10 Richard Bird and Lambert Meertens. Nested Datatypes. In Johan Jeuring, editor, *Mathematics of Program Construction, MPC'98, Proceedings*, volume 1422 of *Lecture Notes in Computer Science*, pages 52–67. Springer, 1998.

11 Richard S. Bird and Ross Paterson. De Bruijn Notation as a Nested Datatype. *J. Funct. Program.*, 9(1):77–91, 1999. URL: `http://journals.cambridge.org/action/displayAbstract?aid=44239`, `doi:10.1017/S0956796899003366`.

12 Jasmin Christian Blanchette, Lorenzo Gheri, Andrei Popescu, and Dmitriy Traytel. Bindings as bounded natural functors. *Proc. ACM Program. Lang.*, 3(POPL):22:1–22:34, 2019. `doi:10.1145/3290335`.

13 Peio Borthelle, Tom Hirschowitz, and Ambroise Lafont. A cellular Howe theorem. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 273–286. ACM, 2020. `doi:10.1145/3373718.3394738`.

14 Matteo Capucci and Bruno Gavranović. Actegories for the working amthematician, 2022. `doi:10.48550/arXiv.2203.16351`.

15 Marcelo Fiore and Dmitrij Szamozvancev. Formal metatheory of second-order abstract syntax. *Proc. ACM Program. Lang.*, 6(POPL):1–29, 2022. `doi:10.1145/3498715`.

16 Marcelo P. Fiore. Second-order and dependently-sorted abstract syntax. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pages 57–68. IEEE Computer Society, 2008. `doi:10.1109/LICS.2008.38`.

17 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. IEEE Computer Society, 1999. `doi:10.1109/LICS.1999.782615`.

18 André Hirschowitz, Tom Hirschowitz, Ambroise Lafont, and Marco Maggesi. Variable binding and substitution for (nameless) dummies. In Patricia Bouyer and Lutz Schröder, editors, *Foundations of Software Science and Computation Structures - 25th International Conference, FOSSACS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings*, volume 13242 of *Lecture Notes in Computer Science*, pages 389–408. Springer, 2022. `doi:10.1007/978-3-030-99253-8_20`.

19 Chung-Kil Hur. *Categorical equational systems : algebraic models and equational reasoning*. PhD thesis, University of Cambridge, UK, 2010. URL: `http://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.608664`.

20 Alexander Kurz, Daniela Petrisan, Paula Severi, and Fer-Jan de Vries. Nominal coalgebraic data types with applications to lambda calculus. *Log. Methods Comput. Sci.*, 9(4), 2013. `doi:10.2168/LMCS-9(4:20)2013`.

21 Thomas Lamiaux and Benedikt Ahrens. An introduction to different approaches to initial semantics, 2024. `arXiv:2401.09366`.

22 Ralph Matthes and Tarmo Uustalu. Substitution in non-wellfounded syntax with variable binding. *Theoretical Computer Science*, 327(1-2):155–174, 2004. `doi:10.1016/j.tcs.2004.07.025`.

23 Ralph Matthes, Kobe Wullaert, and Benedikt Ahrens. Substitution for non-wellfounded syntax with binders through monoidal categories. *CoRR*, abs/2308.05485, 2023. `doi:10.48550/arXiv.2308.05485`, minimally version 3.

24 Paul-André Melliès. Higher-order parity automata. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017. `doi:10.1109/LICS.2017.8005077`.

25 Stefan Milius. Completely iterative algebras and completely iterative monads. *Inf. Comput.*, 196(1):1–41, 2005. doi:10.1016/j.ic.2004.05.003.

26 Andrei Popescu. Nominal recursors as epi-recursors. *Proc. ACM Program. Lang.*, 8(POPL):425–456, 2024. doi:10.1145/3632857.

27 José Espírito Santo, Ralph Matthes, and Luís Pinto. A coinductive approach to proof search through typed lambda-calculi. *Ann. Pure Appl. Log.*, 172(10):103026, 2021. doi:10.1016/j.apal.2021.103026.

28 The Coq Development Team. The Coq proof assistant, version 8.17, 2023. URL: https://zenodo.org/record/8161141.

29 Vladimir Voevodsky, Benedikt Ahrens, Daniel Grayson, et al. UniMath — a computer-checked library of univalent mathematics. Available at http://unimath.github.io/UniMath/ , 2021.

30 Julianna Zsidó. *Typed Abstract Syntax*. PhD thesis, University of Nice Sophia Antipolis, 2010. URL: http://tel.archives-ouvertes.fr/tel-00535944/.

A An Easy Example for the Application Scenario

To ease the understanding of the example given in Section 1, we also show the one of Church numerals.

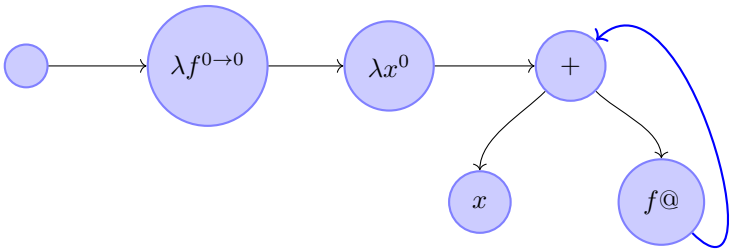


Figure 6 Forest representation of all Church numerals, including infinity.

Let 0 be a base type. We define a closed forest of type $(0 \to 0) \to 0 \to 0$ in Figure 6 [27, Example 5]. $f@$ is short for $f\langle N\rangle$ with N given by where the arrow points to. The back link (in blue and thick) forms a cycle that does not go through a λ-abstraction. Hence, we get a rational tree (with only a finite number of non-isomorphic subtrees). This is a representation of all Church numerals, including infinity, and they are all the "solutions" (including the non-wellfounded ones) for the search for inhabitants in long normal form of the type $(0 \to 0) \to 0 \to 0$. Thus, by infinite unfolding of the just binary choice, an infinite number of finite solutions and even one infinite solution are obtained. The latter is the only infinite Church numeral, obtained by looping with f. In naive proof search, this is at least a potential outcome, and one may want to analyze this phenomenon.

B Recalling Some Notions of the Category-theoretical Background

For reasons of space limitations, this appendix is only available in [23].

C Proof of Theorem 4

This short appendix completes the proof of Theorem 4 in Section 3.2. It also discusses why the result on final coalgebras is in some sense easier than its counterpart for initial algebras.

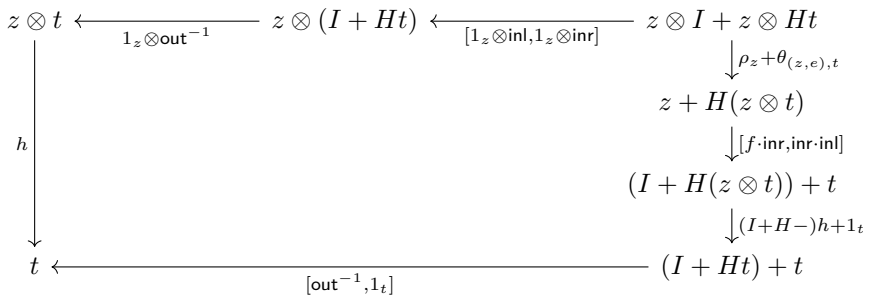


Figure 7 Diagram characterizing a solution h for eqm.

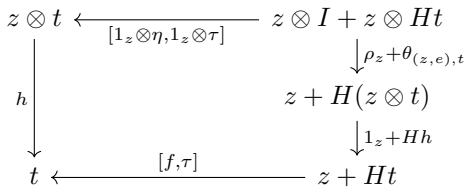


Figure 8 Diagram characterizing morphism h that should be $(\!|f|\!)_{(z,e)}$.

The diagram describing a solution of eqm is given in Figure 7. Of course, the diagram governing $(\!|f|\!)_{(z,e)}$ can be brought into a single equation over coproducts, as seen in Figure 8. The chain of morphisms from $z \otimes I + z \otimes Ht$ to t in both diagrams – Figure 7 and Figure 8 – is identical on the path to the left, as well as on the path to the right.

We remark that Theorem 4 (and also its proof) is slicker than the case of wellfounded syntax studied in [5] (however, concretely for endofunctor categories) where extra requirements beyond being an initial algebra come into play so as to guarantee the applicability of a categorical Mendler-style recursion scheme. This difference can be motivated as follows: substitution for functor T is represented by a monadic multiplication operation of type $T \cdot T \to T$ (with $T \cdot T$ self-composition of T). In the non-wellfounded case, this has the support T of the final coalgebra as target, which is suitable for using finality. However, in the wellfounded case, the source type is $T \cdot T$ and not just T that would be the basis for using initiality.

D Pointed Tensorial Strength for the Signature Functor

For reasons of space limitations, this appendix is only available in [23].

E On the Formalization

Most of the definitions and results presented in this paper are formalized and computer-checked in `UniMath` [29], a library of univalent mathematics based on the computer proof assistant `Coq` [28]. An exception is the application scenario in Section 1; its formalization is ongoing work. For this application, we can only offer the instantiation of the general constructions of this paper but not yet the inhabitation analysis alluded to in Figure 2. Our HTML documentation is derived from commit 7432fee of the `UniMath` library. Proof-checking and creation of the HTML documentation can easily be reproduced at home by following the `UniMath` compilation instructions – do try this at home!

Concerning coinductive definitions, `Coq` features a built-in mechanism for specifying coinductive types (via the keyword `CoInductive`) and for defining functions by corecursion. However, the `UniMath` library departs from standard use of `Coq` in that such declarations of coinductive datatypes are not part of the language used in `UniMath`. Furthermore, definitions by corecursion in `Coq` face numerous issues with guardedness, in particular with so-called "mixed inductive-coinductive" declarations [9] – declarations where the coinductive type makes use of a parameterized inductive type whose parameter is built with the coinductive type. The coinductive calculus of our application scenario (see Section 1), with its lists of alternatives and arguments, falls into that class. In the formalization of the contents of this paper, we therefore construct coinductive datatypes from other type constructors, rather than postulating (a class of) coinductive datatypes using meta-theoretic devices. Our approach is thus comparable to the one employed for working with coinductive datatypes in the Isabelle system [12] and of the construction of indexed M-types in univalent foundations [1]; in all of these cases, a major goal is to keep the "trusted code base" small.

We now discuss some design choices we made in the formalization. When formalizing mathematics in a formal system, some design choices need to be made that are not of mathematical significance: different choices lead to (trivially) equivalent mathematical concepts. Nevertheless, making the right choices can be crucial for the maintainability and usability of the formal library. An example of such a choice is the following. In Section 2.1, we said that a monoidal category is given by a six-tuple, with the tensor component a bifunctor. However, for the sake of our formalization, we have chosen a different but equivalent format to present the tensor operation that we are calling "whiskered". Here, the object mapping of $\otimes$ is replaced by its curried version, and the morphism mapping is replaced by two families of endofunctors on $\mathcal{C}$ that represent the morphism mappings with one of the arguments fixed to the identity morphism – thus the "whiskerings" of that bifunctor. The whiskered definition avoids functors on cartesian products of categories. Such functors do not behave well in practice: the inference of the implicit object arguments – which are pairs of objects – to the functorial action on morphisms often fails, and thus these arguments need to be given explicitly. This would make the formalization cumbersome – which is why we adopted the whiskered format for our work and hence do not suffer from those problems. A third alternative to the traditional definition and the whiskered definition would be a currying of the tensor, to be a functor into a functor category. However, this definition would not provide a clean separation between data and properties – another prerequisite for a library that scales well, in our experience.

On the Logical Structure of Some Maximality and Well-Foundedness Principles Equivalent to Choice Principles

Hugo Herbelin
Université de Paris Cité, Inria, CNRS, IRIF, France

Jad Koleilat
Université Paris Cité, France

Abstract

We study the logical structure of *Teichmüller-Tukey lemma*, a maximality principle equivalent to the axiom of choice and show that it corresponds to the generalisation to arbitrary cardinals of *update induction*, a well-foundedness principle from constructive mathematics classically equivalent to the axiom of dependent choice.

From there, we state general forms of maximality and well-foundedness principles equivalent to the axiom of choice, including a variant of Zorn's lemma. A comparison with the general class of choice and bar induction principles given by Brede and the first author is initiated.

2012 ACM Subject Classification Theory of computation → Proof theory

Keywords and phrases axiom of choice, Teichmüller-Tukey lemma, update induction, constructive reverse mathematics

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.26

1 Introduction

1.1 Context

The axiom of choice is independent of Zermelo-Fraenkel set theory and equivalent to many other formulations [4, 5, 6], the most famous ones being Zorn's lemma, a maximality statement, and Zermelo's theorem, a well-ordering thus also well-foundedness theorem, since well-foundedness and well-ordering are logically dual notions.

In the family of maximality theorems equivalent to the axiom of choice one statement happens to be particularly concise and general, it is Teichmüller-Tukey lemma, that states that every non-empty collection of *finite character*, that is, characterised only by its finite sets, has a maximal element with respect to inclusion.

The axiom of dependent choice is a strict consequence of the axiom of choice. In the context of constructive mathematics, various statements classically but non intuitionistically equivalent to the axiom of dependent choice are considered, such as bar induction, open induction [3], or, more recently, update induction [1], the last two relying on a notion of *open* predicate over functions of countable support expressing that the predicate depends only on finite approximations of the function.

In a first part of the paper, we reason intuitionistically and show that the notion of finite character, when specialised to countable sets, is dual to the notion of open predicate, or, alternatively, that the notion of open predicate, when generalised to arbitrary cardinals is dual to the notion of finite character. As a consequence, we establish that update induction and the specialisation of Teichmüller-Tukey lemma to countable sets are logically dual statements, or, alternatively, that Teichmüller-Tukey lemma and the generalisation of update induction to arbitrary cardinals are logically dual.

In a second part of the paper, we show how Teichmüller-Tukey lemma and Zorn's lemma can be seen as mutual instances the one of the other.

Finally, in a third part, we introduce a slight variant of Teichmüller-Tukey lemma referring to functions rather than sets and make some connections with the classification of choice and bar induction principles studied by Brede and the first author in [2].

The ideas of Section 2 have been developed during an undergraduate internship of the second author under the supervision of the first author in 2022, leading to the idea in Section 4.1 of introducing $\exists\mathbf{MPCF}$ by the second author. Section 3 contains extra investigations made in 2023 by the second author. Section 4.2 contains investigations made jointly in 2024 by the authors.

1.2 The logical system

In this section we describe the logical setting and give definitions that are used throughout the article. The results we prove do not depend greatly on its structure as they require only basic constructions, we shall make precise exactly was is necessary and what is left to the preferences of the reader.

We work in an intuitionistic higher order arithmetic equipped with inductive types like the type with one element ($\mathbb{1}$, $0 : \mathbb{1}$), the type of Boolean values ($\mathbb{B}$, $0_{\mathbb{B}}, 1_{\mathbb{B}} : \mathbb{B}$), the type of natural numbers ($\mathbb{N}$), the product type ($A \times B$), or the coproduct type ($A + B$). In particular, we write $B_\perp$ for the coproduct of B and of $\mathbb{1}$, identifying $b : B$ with $\mathrm{inl}(b) : B_\perp$ and $\perp$ with $\mathrm{inr}(0)$ where inl and inr are the two injections of the coproduct.

We write *Prop* for the type of propositions. For all types A, the type $\mathcal{P}(A)$ denotes the type $A \to \mathit{Prop}$, we shall sometimes refer to it as "subsets of A". We also use the type $\mathbb{N} \to A_\perp$, shortly $A_\perp^{\mathbb{N}}$, to represent the countable subsets of A, implicitly referring to the non-$\perp$ elements of the image of the function[1].

We also require a type for lists: for all types A we denote by A^* the type of lists of terms of type A defined as follows:

$$\frac{}{\varepsilon : A^*} \qquad \frac{u : A^* \quad a : A}{u@a : A^*}$$

We inductively define $\star : A^* \to A^* \to A^*$, the concatenation of two lists:

$$\frac{u : A^*}{u \star \varepsilon := u} \qquad \frac{u : A^* \quad v : A^* \quad a : A}{u \star (v@a) := (u \star v)@a}$$

We denote by $[a_1, \dots, a_n]$ the list $(\dots(\varepsilon@a_1)@\dots)@a_n)$, since $\star$ is associative we drop the parentheses. If $n \in \mathbb{N}$ and $\alpha : A^{\mathbb{N}}$, we write $\alpha_{|n}$ for the recursively defined list $[\alpha(0), \dots, \alpha(n-1)]$. We define $\in : A \to A^* \to \mathit{Prop}$ as: $a \in u := \exists v, w^{A^*},\ v \star [a] \star w = u$.

The symbol $\in$ will be used as defined above and also as a notation for $P(a)$. To be more precise, for all types A, $P : \mathcal{P}(A)$ and $a : A$ we will write $a \in P$ for $P(a)$ and $a \notin P$ for $P(a) \to \perp$. Continuing with the set-like notations, for $P, Q : \mathcal{P}(A)$ we write $P \subseteq Q$ for $\forall a^A, a \in P \to a \in Q$. We require extensional equality for predicates: for all $P, Q : \mathcal{P}(A)$,

[1] For inhabited A, this is intuitionistically equivalent to considering $\mathbb{N} \to A$.

$P = Q \leftrightarrow P \subseteq Q \wedge Q \subseteq P$[2]. The symbol $\subseteq$ will also be used for lists: for all $u, v : A^*$, $u \subseteq v := \forall a^A, a \in u \to a \in v$. Note that equipped with this relation, lists behave more like finite sets than lists. Nevertheless the list structure is not superfluous as will be shown later.

As a convention, we let the scope of quantifiers spans until the end of the sentence, so, for instance, $\forall n, P \to Q$ reads as $\forall n, (P \to Q)$ and similarly for $\exists$.

1.3 Closure operators and partial functions

Let us now define some closure operators and relations on subsets and lists:

▶ **Definition 1.** *Let A be a type, $u : A^*$, $\alpha : \mathcal{P}(A)$, $T : \mathcal{P}(A^*)$, $P : \mathcal{P}(\mathcal{P}(A))$*

$u \subset \alpha \ : Prop$	$\langle T \rangle \ : \mathcal{P}(\mathcal{P}(A))$
$u \subset \alpha \ := \forall a^A, a \in u \to a \in \alpha$	$\langle T \rangle \ := \lambda \alpha^{\mathcal{P}(A)}. \forall u^{A^*}, u \subset \alpha \to u \in T$
	$\langle T \rangle^\circ \ : \mathcal{P}(\mathcal{P}(A))$
	$\langle T \rangle^\circ \ := \lambda \alpha^{\mathcal{P}(A)}. \exists u^{A^*}, u \subset \alpha \wedge u \in T$
$\langle u \rangle \ : \mathcal{P}(A)$	$\lfloor P \rfloor \ : \mathcal{P}(A^*)$
$\langle u \rangle \ := \lambda x^A.\ x \in u$	$\lfloor P \rfloor \ := \lambda u^{A^*}.\ \langle u \rangle \in P$

The symbol $\langle\ \rangle$ is the translation from "the list world" to "the predicate world". More precisely, $\langle u \rangle$ is the canonical way to see a list as a predicate ($u \subset \alpha \leftrightarrow \langle u \rangle \subseteq \alpha$) and $\langle T \rangle$ is an extension of T as a predicate on subsets, $\alpha : \mathcal{P}(A)$ is in $\langle T \rangle$ if and only if it can be arbitrarily approximated by lists of T. Dually, $\lfloor\ \rfloor$ is the translation from predicate to list, taking predicate of finite domain to all lists of elements in the domain. Note that $\langle T \rangle$ is downward closed, that is, $\alpha \subset \beta$ and $\beta \in \langle T \rangle$ implies $\alpha \in \langle T \rangle$. Note also that $\langle \lfloor P \rfloor \rangle$ is a downward closure operator, defining the largest downward closed subset of P. On its side, $\lfloor \langle T \rangle \rfloor$ builds the downward closure up to permutation and replication of the elements of the lists of T. Also, symmetrical properties applies to $\langle\ \rangle^\circ$ exchanging downward with upward and largest subset with smallest superset. Finally, notice that $\langle T \rangle$ may be empty, in fact $\langle T \rangle$ is inhabited if and only if $\varepsilon \in T$, and the same for $\langle T \rangle^\circ$.

Examples

Consider $T : \mathcal{P}(\mathbb{B}^*)$, for simplicity let us use set-like notations when defining T. If $T := \{[1_\mathbb{B}, 0_\mathbb{B}], [1_\mathbb{B}], [0_\mathbb{B}], \epsilon\}$ then $\langle T \rangle$ will contain all subsets of $\mathbb{B}$. Now, if $T := \{[1_\mathbb{B}, 0_\mathbb{B}], [1_\mathbb{B}], [0_\mathbb{B}]\}$, $\langle T \rangle$ will be empty since for all $\alpha : \mathcal{P}(\mathbb{B})$, $\epsilon \subset \alpha$ but $\epsilon \notin T$. If $T := \{\epsilon, [1_\mathbb{B}], [1_\mathbb{B}, 0_\mathbb{B}]\}$ then $\langle T \rangle$ will contain only the empty subset and the singleton containing $1_\mathbb{B}$. Now consider $T' := \{\epsilon, [1_\mathbb{B}], [1_\mathbb{B}, 1_\mathbb{B}], [0_\mathbb{B}, 1_\mathbb{B}], [1_\mathbb{B}, 0_\mathbb{B}, 1_\mathbb{B}, 1_\mathbb{B}]\}$, notice that $\langle T \rangle = \langle T' \rangle$. The $\langle\ \rangle$ operation does not care for duplications or permutations.
For $T := \{\epsilon, [1_\mathbb{B}], [1_\mathbb{B}, 0_\mathbb{B}]\}$, $\lfloor \langle T \rangle \rfloor$ is $\{\epsilon, [1_\mathbb{B}], [1_\mathbb{B}, 1_\mathbb{B}], [1_\mathbb{B}, 1_\mathbb{B}, 1_\mathbb{B}], \dots\}$. Similarly, for $T := \{\epsilon, [1_\mathbb{B}], [0_\mathbb{B}], [1_\mathbb{B}, 0_\mathbb{B}]\}$, $\lfloor \langle T \rangle \rfloor$ is the set of all lists on $\mathbb{B}$.
The $\langle\ \rangle^\circ$ operator has the dual behaviour. Consider $T : \mathcal{P}(\mathbb{N}^*)$, $T := \{[1]\}$ then, $\langle T \rangle^\circ$ contains exactly all subsets of $\mathbb{N}$ containing 1. Similarly if $\epsilon \in T$, then $\langle T \rangle^\circ$ contains all subsets of $\mathbb{N}$. For $T := \{[1]\}$, $\lfloor \langle T \rangle^\circ \rfloor$ will contain every list on $\mathbb{N}$ that contains at least one 1.

We also give similar definitions relatively to countable subsets, abbreviating $(A_\perp)^*$ into $A^*_\perp$:

[2] Extensionality for predicates is assumed for convenience, it is not fundamentally needed

▶ **Definition 2.** *Let A be a type, $u : A^*_\perp$, $\alpha : A^{\mathbb{N}}_\perp$ and $T : \mathcal{P}(A^*_\perp)$*

$$\begin{array}{ll} u \subset_{\mathbb{N}} \alpha \ : \mathit{Prop} & \langle T\rangle_{\mathbb{N}} \ : \mathcal{P}(A^{\mathbb{N}}_\perp) \\ u \subset_{\mathbb{N}} \alpha \ := \exists n^{\mathbb{N}}, u = \alpha_{|n} & \langle T\rangle_{\mathbb{N}} \ := \lambda\alpha^{A^{\mathbb{N}}_\perp}.\forall u^{A^*}, u \subset_{\mathbb{N}} \alpha \to u \in T \\ & \langle T\rangle^\circ_{\mathbb{N}} \ : \mathcal{P}(A^{\mathbb{N}}_\perp) \\ & \langle T\rangle^\circ_{\mathbb{N}} \ := \lambda\alpha^{A^{\mathbb{N}}_\perp}.\exists u^{A^*}, u \subset_{\mathbb{N}} \alpha \wedge u \in T \end{array}$$

We conclude this section defining two different notions of partial functions:

▶ **Definition 3** (Relational partial function)**.** *Let A, B be types, a relational partial function f from A to B is a relational functional relation of $\mathcal{P}(A \times B)$. Formally, a relational partial function from A to B is a term $f : \mathcal{P}(A \times B)$ such that $\forall a^A, \forall b, b'^B,\ ((a,b) \in f \wedge (a,b') \in f) \to b = b'$. Its domain is defined by:*

$$\begin{array}{l} \mathbf{dom}(f) : \mathcal{P}(A) \\ \mathbf{dom}(f) := \lambda a^A.\exists b^B, (a,b) \in f \end{array}$$

For all $a' : A$, we denote by $\mathbf{dom}(f) \cup a'$ the predicate $\lambda a^A.(\exists b^B, (a,b) \in f) \vee a = a'$.

▶ **Definition 4** (Decidable partial function)**.** *Let A, B be types, a decidable partial function f from A to B is a total function $f : A \to B_\perp$. Its domain and graph are defined by:*

$$\begin{array}{ll} \mathbf{dom}(f) : \mathcal{P}(A) & \mathcal{G}(f) : \mathcal{P}(A \times B) \\ \mathbf{dom}(f) := \lambda a^A.f(a) \neq \perp & \mathcal{G}(f) := \lambda(a,b)^{A\times B}.f(a) = \mathrm{inl}(b) \end{array}$$

For all $a' : A$, we denote by $\mathbf{dom}(f) \cup a'$ the predicate $\lambda a^A.\ f(a) \neq \perp \vee a = a'$.

Notation

We write $f \in A \to_{\mathrm{p}} B$ to denote that f is a relational partial function from A to B and $f : A \to B_\perp$ for the type of decidable partial functions from A to B. We will also write $\Theta f^{A\to_{\mathrm{p}} B}, P$ for $\Theta f^{\mathcal{P}(A\times B)}, (f \in A \to_{\mathrm{p}} B) \to P$ for $\Theta \in \{\lambda, \forall, \exists\}$.

The difference between these two definitions is in the decidability of the domain: decidable partial functions have a decidable domain while it's not the case of relational partial functions. The graph operation $\mathcal{G}$ allows us to recover a relational partial function from a decidable partial function. One needs excluded middle to recover a decidable partial function from a relation partial function, hence decidable partial functions are stronger axiomatically. Notice that we used the same notation **dom** in both definitions. Since they both have the same semantic meaning and we will make clear whether we are using relation partial function or decidable partial function, it should not cause any confusion.

2 TTL and UI

In this section, we define Teichmüller-Tukey lemma and update induction and emphasise that they are logically dual, up to the difference that the former is relative to predicates over subsets of arbitrary cardinals while update induction is relative to predicates over countable subsets. Underneath, they rely on the dual notions of predicate of finite character and of open predicate.

2.1 Predicates of finite character

A set is of *finite character* if all its information is contained in its finite elements. In our setting, a predicate $P : \mathcal{P}(\mathcal{P}(A))$ is of finite character if all its information is contained in a predicate over lists. There are two canonical ways to express this:

▶ **Definition 5** (Finite character). *Let A be a type and $P : \mathcal{P}(\mathcal{P}(A))$. We propose two definitions of finite character:*

$$P \in \mathbf{FC}_1 := \forall \alpha^{\mathcal{P}(A)}, \alpha \in P \leftrightarrow \forall u^{A^*}, u \subset \alpha \to u \in \lfloor P \rfloor$$
$$P \in \mathbf{FC}_2 := \exists T^{\mathcal{P}(A^*)}, \langle T \rangle = P$$

Rewriting $\mathbf{FC}_1$ using the terms just defined:

$$P \in \mathbf{FC}_1 := P = \langle \lfloor P \rfloor \rangle$$

$\mathbf{FC}_1$ and $\mathbf{FC}_2$ are, in essence, paraphrases of one an other, thus there is no reason not to expect them to be equivalent. First we will need a lemma:

▶ **Lemma 6.** *Let A be a type and $T : \mathcal{P}(A^*)$ then $\langle T \rangle \in \mathbf{FC}_1$.*

Proof. Let $\alpha : \mathcal{P}(A)$. Suppose $\alpha \in \langle T \rangle$, our goal is to show that $\alpha \in \langle \lfloor \langle T \rangle \rfloor \rangle$. Let $u : A^*$ such that $u \subset \alpha$, we will show that $u \in \lfloor \langle T \rangle \rfloor$. By definition $u \in \lfloor \langle T \rangle \rfloor$ if and only if $\langle u \rangle \in \langle T \rangle$ if and only if every sublist of u is in T. Since α can be arbitrarily approximated by terms of T and $u \subset \alpha$, so can u. Hence, $u \in \lfloor \langle T \rangle \rfloor$ thus, $\alpha \in \langle \lfloor \langle T \rangle \rfloor \rangle$.
Suppose $\alpha \in \langle \lfloor \langle T \rangle \rfloor \rangle$, then for all $u : A^*$ such that $u \subset \alpha$, $u \in \lfloor \langle T \rangle \rfloor$ which we can rewrite as $\langle u \rangle \in \langle T \rangle$. We easily show that $\langle u \rangle \in \langle T \rangle \to u \in T$ thus $\alpha \in \langle T \rangle$. ◀

We have shown that $\langle T \rangle = \langle \lfloor \langle T \rangle \rfloor \rangle$. This means that without loss of generality, we can require in $\mathbf{FC}_2$ that the witness T be of the form $\lfloor \langle T' \rangle \rfloor$ for some T'. This is a way to express that T can be chosen to be minimal. In fact if we are given P and T such as in $\mathbf{FC}_2$, it may happen that T contains a list u that is not closed under $\subseteq$ (i.e.. $v \subseteq u \not\to v \in T$). Such an u will be invisible when looking at $\langle T \rangle$, hence we can consider u as a superfluous term. The $\lfloor \langle\ \rangle \rfloor$ operation allows us, without loss of generality, to remove those terms, thus making T minimal.

▶ **Theorem 7.** $\mathbf{FC}_1 \leftrightarrow \mathbf{FC}_2$

Proof. Let A be a type and $P : \mathcal{P}(\mathcal{P}(A))$. From left to right: suppose $P \in \mathbf{FC}_1$. $\lfloor P \rfloor$ is a witness of $P \in \mathbf{FC}_2$.
From right to left: suppose $P \in \mathbf{FC}_2$, let T be the witness of $P \in \mathbf{FC}_2$. By lemma 6 $\langle \lfloor \langle T \rangle \rfloor \rangle = \langle T \rangle$ and by hypothesis $P = \langle T \rangle$, we can rewrite the first equality as $\langle \lfloor P \rfloor \rangle = P$. ◀

Since $\mathbf{FC}_1$ and $\mathbf{FC}_2$ are equivalent, we will from now on write $\mathbf{FC}$ without the indices.

2.2 Open predicates

A notion of *open* predicates over functions of countable domain was defined in Coquand [3] and generalised by Berger [1]. Using the definitions of Section 1.3, a predicate over $\alpha : A^{\mathbb{N}}$ is *open* in the sense of Berger if it has the form $\alpha \in \langle T \rangle_{\mathbb{N}} \to \alpha \in \langle U \rangle^{\circ}_{\mathbb{N}}$ for some $T, U : \mathcal{P}(A^*)$. In order to get a closer correspondence with the notion of finite character, we will however stick to Coquand's definition. Additionally, to get a closer correspondence with the case of open predicates used in update induction, we consider open predicates for functions to $A_\perp$.

▶ **Definition 8** (Countably-open predicate, in Coquand's sense, with partiality). *Let A be a type and $P : \mathcal{P}(A^{\mathbb{N}}_{\perp})$. We define:*

$$P \in \mathbf{OPEN}_{\mathbb{N}} := \exists T^{\mathcal{P}(A^*_{\perp})}, \langle T \rangle^{\circ}_{\mathbb{N}} = P$$

The observations made on predicates of finite character apply to countably-open predicates, namely that $\langle T \rangle^{\circ}_{\mathbb{N}} = \left\langle \left\lfloor \langle T \rangle^{\circ}_{\mathbb{N}} \right\rfloor \right\rangle^{\circ}_{\mathbb{N}}$. Obviously, we can also move from $A^{\mathbb{N}}_{\perp}$ to $\mathcal{P}(A)$ and introduce a general notion of open predicates which again, will satisfy $\langle T \rangle^{\circ} = \left\langle \left\lfloor \langle T \rangle^{\circ} \right\rfloor \right\rangle^{\circ}$:

▶ **Definition 9** (Open predicate). *Let A be a type and $P : \mathcal{P}(\mathcal{P}(A))$. We define:*

$$P \in \mathbf{OPEN} := \exists T^{\mathcal{P}(A^*)}, \langle T \rangle^{\circ} = P$$

Conversely, we can define a notion of predicate of countably-finite character dual the notion of countably-open predicate:

▶ **Definition 10** (Predicate of countably-finite character). *Let A be a type and $P : \mathcal{P}(A^{\mathbb{N}}_{\perp})$. We define:*

$$P \in \mathbf{FC}_{\mathbb{N}} := \exists T^{\mathcal{P}(A^*_{\perp})}, \langle T \rangle_{\mathbb{N}} = P$$

This finally results in the following dualities:

Table 1 Predicates of finite character VS Open predicate.

	Universal notion	Existential notion
Arbitrary subsets	Finite character	Open
Countable subsets	Countably-finite character	Countably-open

2.3 Teichmüller-Tukey lemma and Update induction

Before defining Teichmüller-Tukey lemma we need a few definitions:

▶ **Definition 11.** *Let A be a type, $P : \mathcal{P}(\mathcal{P}(A))$ and $\alpha, \beta : \mathcal{P}(A)$. We define:*

$$\beta \prec \alpha : \mathit{Prop}$$
$$\beta \prec \alpha := \exists a^A, a \notin \alpha \wedge \beta = (\lambda x^A.\ x \in \alpha \vee x = a)$$
$$\alpha \in \mathbf{Max}_{\prec}(P) : \mathit{Prop}$$
$$\alpha \in \mathbf{Max}_{\prec}(P) := \alpha \in P \wedge \forall \beta^{\mathcal{P}(A)}, \beta \prec \alpha \to \beta \notin P$$

Thus, $\beta \prec \alpha$ stands for β extends α (if β is an update of α) while $\mathbf{Max}_{\prec}(P)$ is the predicate of maximal elements of $(P, \succ)$ ($\succ$ is the reverse of $\prec$).

What we are interested in are predicates of finite character but Theorem 7 allows us to consider only predicates on lists since there is a correspondence between them. Hence, we will quantify or instantiate schemas on predicate on lists.

▶ **Definition 12** (Teichmüller-Tukey lemma). *Let A be a type and $T : \mathcal{P}(A^*)$, we define the Teichmüller-Tukey lemma,* $\mathbf{TTL}_{AT}$*:*

$$(\exists \alpha^{\mathcal{P}(A)}, \alpha \in \langle T \rangle) \to \exists \alpha^{\mathcal{P}(A)},\ \alpha \in \mathbf{Max}_{\prec}(\langle T \rangle)$$

Notations

TTL denotes the full schema: for all types A and all $T : \mathcal{P}(A^*)$, $\exists\alpha^{\mathcal{P}(A)}, \alpha \in \langle T\rangle \to \exists\alpha^{\mathcal{P}(A)}, \alpha \in \mathbf{Max}_{\prec}(\langle T\rangle)$.
$\mathbf{TTL}_{AT}$ denotes the schema specialised in this A and this T.
$\overline{\mathbf{TTL}_{AT}}$ denotes the restriction of the full schema **TTL** to A and T of a particular shape. For example: $\overline{\mathbf{TTL}_{\mathbb{N}T}}$ is the schema: for all $T : \mathcal{P}(\mathbb{N}^*)$, $\exists\alpha^{\mathcal{P}(\mathbb{N})}, \alpha \in \langle T\rangle \to \exists\alpha^{\mathcal{P}(\mathbb{N})}, \alpha \in \mathbf{Max}_{\prec}(\langle T\rangle)$. Moreover, if C_A denotes a particular collection of predicates over lists of A (A is a parameter), then $\overline{\mathbf{TTL}_{AC_A}}$ denotes the restrictions of the schema **TTL** to any A type and $T : \mathcal{P}(A^*)$ that is in C_A.

Following an earlier remark, we impose that the finite character predicate we are considering must be inhabited, without this **TTL** becomes trivially inconsistent. Having defined **TTL** we now show that we can recover an induction principle by using contraposition and Morgan's rules:

Unfolding some definitions, $\mathbf{TTL}_{AT}$ is

$$(\exists\alpha^{\mathcal{P}(A)}, \alpha \in \langle T\rangle) \to \exists\alpha^{\mathcal{P}(A)}, \alpha \in \langle T\rangle \wedge (\forall\beta^{\mathcal{P}(A)}, \beta \prec \alpha \to \beta \notin \langle T\rangle)$$

Contraposing and pushing some negations:

$$\forall\alpha^{\mathcal{P}(A)}, [\neg(\alpha \in \langle T\rangle) \vee \neg\forall\beta^{\mathcal{P}(A)}, \beta \prec \alpha \to \beta \notin \langle T\rangle] \to \forall\alpha^{\mathcal{P}(A)}, \alpha \notin \langle T\rangle$$

We have a sub-formula of the form $\neg A \vee \neg B$, we have the choice of writing it as $A \to \neg B$ or $B \to \neg A$. The first choice leads to a principle we will call $\mathbf{TTL}^{\text{co}}_{AT}$:

$$\forall\alpha^{\mathcal{P}(A)}, [\alpha \in \langle T\rangle \to \exists\beta^{\mathcal{P}(A)}, \beta \prec \alpha \wedge \beta \in \langle T\rangle] \to \forall\alpha^{\mathcal{P}(A)}, \alpha \notin \langle T\rangle$$

And the second choice leads to an induction principle:

$$\forall\alpha^{\mathcal{P}(A)}, [(\forall\beta^{\mathcal{P}(A)}, \beta \prec \alpha \to \beta \notin \langle T\rangle) \to \alpha \notin \langle T\rangle] \to \forall\alpha^{\mathcal{P}(A)}, \alpha \notin \langle T\rangle$$

$\mathbf{TTL}^{\text{co}}$ is intuitively an opposite formulation of **TTL**. The induction principle we obtain seems to express something different. We can push further the negations in order to obtain a positive formulation of it:

$$\forall\alpha^{\mathcal{P}(A)}, [(\forall\beta^{\mathcal{P}(A)}, \beta \prec \alpha \to \beta \in \langle T\rangle^\circ) \to \alpha \in \langle T\rangle^\circ] \to \forall\alpha^{\mathcal{P}(A)}, \alpha \in \langle T\rangle^\circ$$

And this can be seen as as a generalisation of Berger's update induction [1] going from countably-open predicates to arbitrary open predicates.

To state update induction, we need to focus on partial functions from $\mathbb{N}$ to A which we equip with an order:

▶ **Definition 13.** *Let A be a type, $P : \mathcal{P}(A^{\mathbb{N}}_{\perp})$ and $\alpha, \beta : A^{\mathbb{N}}_{\perp}$. We define:*

$$\beta \prec_N \alpha : \mathit{Prop}$$
$$\beta \prec_N \alpha := \exists m^{\mathbb{N}}, \exists a^{A}, \alpha(m) = \perp \wedge \beta(m) = a \wedge \forall n^{\mathbb{N}}, n \neq m \to \alpha(n) = \beta(n)$$

Like **TTL**, update induction is originally defined on open predicates but since any open predicate comes from a predicate on lists, we can more primitively state it as follows:

▶ **Definition 14** (Update induction). *Let A be a type and $T : \mathcal{P}(A^*_\perp)$, we define Update induction,* $\mathbf{UI}_{AT}$*:*

$$\forall \alpha^{A^{\mathbb{N}}_\perp}, [(\forall \beta^{A^{\mathbb{N}}_\perp}, \beta \prec_N \alpha \to \beta \in \langle T \rangle^\circ_{\mathbb{N}}) \to \alpha \in \langle T \rangle^\circ_{\mathbb{N}}] \to \forall \alpha^{A^{\mathbb{N}}_\perp}, \alpha \in \langle T \rangle^\circ_{\mathbb{N}}$$

Contrastingly, we now formally state the dual of **TTL** that we obtained above:

▶ **Definition 15** (Generalised update induction). *Let A be a type and $T : \mathcal{P}(A^*)$, we define Generalised update induction,* $\mathbf{GUI}_{AT}$*:*

$$\forall \alpha^{\mathcal{P}(A)}, [(\forall \beta^{\mathcal{P}(A)}, \beta \prec \alpha \to \beta \in \langle T \rangle^\circ) \to \alpha \in \langle T \rangle^\circ] \to \forall \alpha^{\mathcal{P}(A)}, \alpha \in \langle T \rangle^\circ$$

Also, we introduce a countable version of **TTL**, logically dual to **UI**:

▶ **Definition 16** (Countable Teichmüller-Tukey lemma). *Let A be a type and $T : \mathcal{P}(A^*_\perp)$, we define the countable Teichmüller-Tukey lemma,* $\mathbf{TTL}^{\mathbb{N}}_{AT}$*:*

$$(\exists \alpha^{A^{\mathbb{N}}_\perp}, \alpha \in \langle T \rangle_{\mathbb{N}}) \to \exists \alpha^{A^{\mathbb{N}}_\perp}, \alpha \in \mathbf{Max}_{\prec_{\mathbb{N}}}(\langle T \rangle_{\mathbb{N}})$$

We thus obtain the following table:

Table 2 Maximality principles VS Induction principles.

	Finite character	Open
Arbitrary subsets	$\mathbf{TTL}_{AT}$	$\mathbf{GUI}_{AT}$
Countable subsets	$\mathbf{TTL}^{\mathbb{N}}_{AT}$	$\mathbf{UI}_{AT}$

In particular, since **TTL** is classically equivalent to the full axiom of choice, **GUI** is also classically equivalent to the full axiom of choice.

3 TTL and Zorn's lemma

In this section we analyse precisely the relationships of **TTL** with Zorn's lemma. We go further than showing their equivalence, we look at which part of **TTL** (as a schema) is necessary to prove Zorn's lemma and reciprocally. This equivalence result is also a proof that the version of Teichmüller-Tukey lemma we defined captures the full choice.

▶ **Definition 17.** *Let A be a type, $<$ a strict order on A, $a : A$ and $E, F : \mathcal{P}(A)$. Define:*

$$E \in \mathbf{Ch}(A) : \mathit{Prop}$$
$$E \in \mathbf{Ch}(A) := \forall a, b^A, a, b \in E \to (a < b \vee b < a \vee a = b)$$

$$F \in \mathbf{SCh}(E) : \mathit{Prop}$$
$$F \in \mathbf{SCh}(E) : F \subseteq E \wedge F \in \mathbf{Ch}(A)$$

$$E \in \mathbf{Ind}(A) : \mathit{Prop}$$
$$E \in \mathbf{Ind}(A) := (\forall F^{\mathcal{P}(A)}, F \in \mathbf{SCh}(E) \to \exists a^A, a \in E \wedge \forall b^A, b \in F \to b \leq a)$$

$$a \in \mathbf{Max}_<(E) : \mathit{Prop}$$
$$a \in \mathbf{Max}_<(E) := a \in E \wedge \forall b^A, a < b \to b \notin E$$

Where $\leq$ is the reflexive closure of $<$.

Ch is the chain predicate, **SCh** is the subchain predicate, **Ind** is the inductive "subset" predicate and $\mathbf{Max}_<$ is simply the maximal element predicate. We choose to express these definitions in terms of predicates over types rather than directly in terms of types, to avoid discussions on proof relevance and stay in a more general setting. If we were proof-irrelevant, instantiating our schemas on predicates over types would be identical to doing it directly on types which would simplify notations and yield the same results.
We can now define concisely Zorn's lemma:

▶ **Definition 18** (Zorn lemma). *Let A be a type, $<$ a strict order on A, and E a predicate on A.* $\mathbf{Zorn}_{A<E}$ *is the following statement*

$$E \in \mathbf{Ind} \to \exists a^A, a \in \mathbf{Max}_<(E)$$

▶ **Theorem 19.** $\mathbf{TTL} \leftrightarrow \mathbf{Zorn}$

The following is an adaptation of a usual set-theoretic proof in our setting.

Proof. From left to right: fix A a type, $<$ a strict order on A and $E : \mathcal{P}(A)$ such that $E \in \mathbf{Ind}(A)$. We first show that $\mathbf{SCh}(E)$ is of finite character:

Let $F : \mathcal{P}(A)$ such that $F \in \mathbf{SCh}(E)$, we show $F \in \langle \lfloor \mathbf{SCh}(E) \rfloor \rangle$: let $u : A^*$ such that $u \subset F$, $\langle u \rangle$ is thus a chain of E therefore $u \in \lfloor \mathbf{Ch}(E) \rfloor$. Let $F : \mathcal{P}(A)$ such that $F \in \langle \lfloor \mathbf{SCh}(E) \rfloor \rangle$, by choosing lists of length 2 we can show that F is a subchain of E. Hence $\mathbf{SCh}(E) \in \mathbf{FC}$.

Using $\mathbf{TTL}_{A\lfloor \mathbf{SCh}(E) \rfloor}$, we get $G : \mathcal{P}(A)$ such that $G \in \mathbf{Max}(\mathbf{SCh}(E))$. G is a subchain of E, since E is inductive we get $g : A$ such that $g \in E$ and $\forall a A, a \in G \to a < g$. Suppose we have $h : A$ such that $g < h$ and $h \in E$. Let $G' := \lambda a^A.a \in G \vee a = h$, then we have $G' \prec G$, since $G \in \mathbf{Max}(\mathbf{SCh}(E))$, $G' \notin \mathbf{SCh}(E)$. On the other side, G' is obviously a chain and $G' \subseteq E$, therefore $G' \in \mathbf{SCh}(E)$. This is a contradiction, hence $g \in \mathbf{Max}_<(E)$.

From right to left: let $T : \mathcal{P}(A^*)$. $\subset$ is a strict order on $\mathcal{P}(\mathcal{P}(A))$. Since $\langle T \rangle$ is of finite character, a maximal element for $\subset$ is also a maximal element for $\succ$. Hence, what is left to show is that $\langle T \rangle$ is inductive and use $\mathbf{Zorn}_{\mathcal{P}(A)\subset\langle T\rangle}$ to produce a maximal term. Let $Q : \mathcal{P}(\mathcal{P}(A))$ such that $Q \in \mathbf{SCh}(\langle T \rangle)$. Let $\alpha := \lambda a^A.\exists \beta^{\mathcal{P}(A)}, \beta \in Q \wedge a \in \beta$. By construction, α is an upper bound of Q, let's show that it is indeed in $\langle T \rangle$. Since $\langle T \rangle$ is of finite character it suffices to show that for all $u : A^*$, $u \subset \alpha \to u \in T$. Let $u : A^*$ such that $u \subset \alpha$. Since u is a finite list, we can enumerate its elements $a_0, \ldots, a_n$. For all $0 \leq i \leq n$, let $\beta_i : \mathcal{P}(A)$ be such that $a_i \in \beta_i$ and $\beta_i \in Q$. Since Q is chain, there exists $0 \leq i_0 \leq n$ such that for all $0 \leq i \leq n, \beta_i \subseteq \beta_{i_0}$. Thus, $u \subset \beta_{i_0}$, $\beta_{i_0} \in \langle T \rangle$ and so $u \in \langle T \rangle$. ◀

Looking more closely at this proof we notice that we have proved a finer result than simply the equivalence. We have shown $\mathbf{TTL}_{A\lfloor \mathbf{SCh}(E) \rfloor} \to \mathbf{Zorn}_{A<E}$ and $\mathbf{Zorn}_{\mathcal{P}(A)\subset\langle T\rangle} \to \mathbf{TTL}_{AT}$. We can express for a predicate over lists to be of the form $\lfloor \mathbf{SCh}(E) \rfloor$ in a more syntactic way.

▶ **Definition 20.** *Let A be a type and $T : \mathcal{P}(A^*)$, we say that T is a list of chains, if there exists T' such that:*

- $\epsilon \in T'$
- $u@a \in T'$ *and* $[a] \star v \in T'$ *if and only if* $u \star [a] \star v \in T'$
- $u \star [a] \star v \in T'$ *implies* $u \star v \in T'$
- *if* $a \neq b$ *and* $u \star [a] \star v \star [b] \star w \in T'$ *then for all* $u', v', w' : A^*$, $u' \star [b] \star v' \star [a] \star w' \notin T'$

and T is the downward closure of T' by $\subseteq$. We denote by $\mathbf{C}_A$ the collection of lists of chains of A.

▶ **Lemma 21.** *Let A be a type, $<$ a strict order on A and $E : \mathcal{P}(A)$, then there exists $T \in \mathbf{C}_A$ such that $\mathbf{SCh}(E) = \langle T \rangle$. Reciprocally, let A be a type, then for every $T \in \mathbf{C}_A$ there exist a strict order $<$ on A and $E : \mathcal{P}(A)$ such that $\mathbf{SCh}(E) = \langle T \rangle$.*

Proof. Proof of the first statement: we inductively define a $T' : \mathcal{P}(A^*)$.

$$\frac{}{\varepsilon \in T'} \qquad \frac{a \in E}{[a] \in T'} \qquad \frac{b \in E \quad a < b \quad u@a \in T'}{u@a@b \in T'}$$

We easily show that T' satisfies the conditions of the above definition. Let T be the downward closure of T'. Let $F \in \mathbf{SCh}(E)$ and $u : A^*$ such that $u \subset F$. Since F is a chain we can construct a list u' of all elements of u such that u' does not contain twice the same element and is ordered increasingly relative to $<$. u' is thus in T' hence u is in T. Let $F \in \langle T \rangle$ and $a, b : A$ such that $a, b \in F$. By hypothesis the list $[a, b]$ is in T. There exists $u \in T'$ such that $[a, b] \subset u$. Hence $a, b \in \langle u \rangle$ which is a chain. In conclusion F is a subchain of E.

Proof of the reciprocal: suppose given a type A with decidable equality and $T \in \mathbf{C}_A$. There exists a T' satisfying the aforementioned conditions. Let $E := \lambda a^A . \exists u^{A^*}, u \in T' \wedge a \in u$. We now must define an ordering on A. Define $<$ a binary relation on A such that $a < b := [a, b] \in T'$. Using last "axiom" of the definition of T' we easily show that it is irreflexive. For transitivity notice that if $[a, b], [b, c] \in T'$ then $[a, b, c] \in T'$ then $[a, c] \in T'$. Thus, it is a strict ordering on A. Let $F \in \mathbf{SCh}(E)$ and $u : A^*$ such that $u \subset F$. We can assume that u is sorted increasingly relatively to $<$. Using the same trick used for proving transitivity show that $u \in T$. Let $F \in \langle T \rangle$ and $a, b : A$ such that $a, b \in F$. By hypothesis the list $[a, b]$ is in T therefore, $a < b$ which means that F is indeed a chain. ◀

▶ **Corollary 22.** $\overline{\mathbf{TTL}_{A\mathbf{C}_A}} \to \mathbf{Zorn}$ *and* $\overline{\mathbf{Zorn}_{\mathcal{P}(A)\subset\langle T\rangle}} \to \mathbf{TTL}$. *Hence we deduce the somewhat surprising results* $\mathbf{TTL} \leftrightarrow \overline{\mathbf{TTL}_{A\mathbf{C}_A}}$ *and* $\mathbf{Zorn} \leftrightarrow \overline{\mathbf{Zorn}_{\mathcal{P}(A)\subset\langle T\rangle}}$.

Looking back at the path we took to arrive at this conclusion, the results are quite expected, but looking only at the definition of a list of chains it is quite surprising that restricting **TTL** this much does not change its power.

4 ∃MPCF

In this section we define a choice principle ∃**MPCF** which stands for "Exists a Maximal Partial Choice Function" and a weaker version ∃$\mathbf{MPCF}^-$. It is weaker in the sense that ∃**MPCF** implies ∃$\mathbf{MPCF}^-$ but the equivalence is true if we allow excluded middle. We show that ∃$\mathbf{MPCF}^-$ is equivalent in its general form to **TTL** and link ∃**MPCF** to the general class of dependent choice **GDC**, given by Brede and the first author in [2]. In particular, ∃**MPCF** and ∃$\mathbf{MPCF}^-$ can be seen as refinements of **TTL** whose strength is more explicitly controlled.

▶ **Definition 23.** *Let A, B be types, $f, g \in A \to_{\mathrm{p}} B$ and $P : \mathcal{P}(\mathcal{P}(A \times B))$, define:*

$$\begin{aligned}
&g \prec f : \mathit{Prop}\\
&g \prec f : \exists a^A, a \notin \mathbf{dom}(f) \wedge (\mathbf{dom}(g) = \mathbf{dom}(f) \cup a) \wedge\\
&\qquad (\forall x^A, x \in \mathbf{dom}(f) \to \exists b^B, (x, b) \in f \wedge (x, b) \in g)\\
f \in \mathbf{Max_{rpf}}(P) &: \mathit{Prop}\\
f \in \mathbf{Max_{rpf}}(P) &:= f \in P \wedge \forall g^{A \to_{\mathrm{p}} B}, g \prec f \to g \notin P
\end{aligned}$$

▶ **Definition 24** (**∃MPCF$^-$**). *Let A, B be types and $T : \mathcal{P}((A \times B)^*)$, $\exists\mathbf{MPCF}^-_{ABT}$ is the statement:*

$$(\exists \alpha^{\mathcal{P}(A\times B)}, \alpha \in \langle T\rangle) \to \exists f^{A\to_{\mathrm{p}} B}, f \in \mathbf{Max_{rpf}}(\langle T\rangle)$$

▶ **Definition 25.** *Let A, B be types, $f, g : A \to B_\perp$ and $P : \mathcal{P}(\mathcal{P}(A \times B))$, define:*

$$\begin{aligned} &g \prec f : \mathit{Prop}\\ &g \prec f : \exists a^A, a \notin \mathbf{dom}(f) \wedge (\mathbf{dom}(g) = \mathbf{dom}(f) \cup a) \wedge\\ &\qquad (\forall x^A, x \in \mathbf{dom}(f) \to f(x) = g(x))\\ &f \in \mathbf{Max_{dpf}}(P) : \mathit{Prop}\\ &f \in \mathbf{Max_{dpf}}(P) := \mathcal{G}(f) \in P \wedge \forall g^{A\to B_\perp}, g \prec f \to \mathcal{G}(g) \notin P \end{aligned}$$

Since the intuitive meaning is the same we use the symbol $\prec$ for predicate, for relational partial functions and decidable partial function.

▶ **Definition 26** (**∃MPCF**). *Let A, B be types and $T : \mathcal{P}((A \times B)^*)$, the theorem of existence of a maximal partial choice function $\exists\mathbf{MPCF}_{ABT}$ is the statement:*

$$(\exists \alpha^{\mathcal{P}(A\times B)}, \alpha \in \langle T\rangle) \to \exists f^{A\to B_\perp}, f \in \mathbf{Max_{dpf}}(\langle T\rangle)$$

The difference between **∃MPCF** and **∃MPCF$^-$** lies solely in the "kind" of partial function that is used. Hence, as per the above remark on the differences between relation partial function and decidable partial function, **∃MPCF** $\to$ **∃MPCF$^-$** and assuming excluded middle **∃MPCF$^-$** $\to$ **∃MPCF** which we denote by **∃MPCF$^-$** $\to_{\mathrm{cl}}$ **∃MPCF**.

4.1 ∃MPCF and TTL

Now that we have defined **∃MPCF$^-$**, we show that it is equivalent to **TTL** hence, **∃MPCF** $\to$ **TTL** and **TTL** $\to_{\mathrm{cl}}$ **∃MPCF**.

▶ **Theorem 27.** *Let A be a type, $T : \mathcal{P}(A^*)$ and π^*T the operation that maps T to $\lambda u^{(A\times\mathbb{1})^*}.\ \pi(u) \in T$ where π is the canonical projection of $(A \times \mathbb{1})^*$ on A^*. Then, $\exists\mathbf{MPCF}^-_{A\mathbb{1}\pi^*T} \to \mathbf{TTL}_{AT}$. Let A, B be types and $T : \mathcal{P}((A \times B)^*)$ then, $\mathbf{TTL}_{(A\times B)T} \to \exists\mathbf{MPCF}^-_{ABT}$.*

Proof. **∃MPCF$^-_{A\mathbb{1}\pi^*T}$** $\to$ **TTL$_{AT}$**: let A a type, $T : \mathcal{P}(A^*)$ and $\pi^*T := \lambda u^{(A\times\mathbb{1})^*}.\ \pi(u) \in T$. From **∃MPCF$^-_{A\mathbb{1}\pi^*T}$** we obtain $f \in A \to_{\mathrm{p}} \mathbb{1}$ such that $f \in \mathbf{Max_{rpf}}(\langle \pi^*T\rangle)$. Define $\alpha := \mathbf{dom}(f)$ and let's show that $\alpha \in \mathbf{Max}(\langle T\rangle)$. By construction, α is in $\langle T\rangle$. Suppose $\beta : \mathcal{P}(A \times B)$ such that $\beta \prec \alpha$. We can construct a relational partial function $g : A \to_{\mathrm{p}} \mathbb{1}$ such that $\beta = \mathbf{dom}(g)$. Since $g \prec f$, g is not in $\langle U\rangle$ hence β is not in $\langle T\rangle$.

TTL$_{(A\times B)T}$ $\to$ **∃MPCF$^-_{ABT}$**: let A, B types and $T : \mathcal{P}((A \times B)^*)$. Define

$$Q := \lambda u^{(A\times B)^*}.\ (\forall a^A, \forall b, b'^B, (a, b) \in u \wedge (a, b') \in u \to b = b') \wedge u \in T$$

Notice that $\langle Q\rangle$ is not empty, since $\langle T\rangle$ is inhabited, $\epsilon \in T$. From this, we deduce that $\epsilon \in Q$ hence, the empty predicate is in $\langle Q\rangle$. We can now apply **TTL$_{(A\times B)Q}$** and get α such that $\alpha \in \mathbf{Max}(\langle Q\rangle)$. By construction α is a relational partial function. It follows that it's a maximal relational partial function, thus proving **∃MPCF$^-_{ABT}$**. ◀

TTL can be seen as a projection of **∃MPCF**. The fact that they are so tightly linked is not surprising as "being a partial function" for a subset of $A \times B$ is a property of finite character.

4.2 ∃MPCF and GDC

Introduced in [2], Generalised Dependent Choice ($\mathbf{GDC}_{ABT}$) is a common generalisation of the axiom of dependent choice and of the Boolean prime ideal theorem. Parameterised by a domain A, a codomain B and a predicate $T : \mathcal{P}((A \times B)^*)$, it yields dependent choice when A is countable, the Boolean prime ideal theorem when B is two-valued, and the full axiom of choice when T comes as the "alignment" of some relation (see below). To the difference of ∃**MPCF**, **GDC** asserts the existence of a total choice function, but this to the extra condition of a property of "approximability" of T by arbitrary long finite approximations. To the difference of ∃**MPCF** whose strength is the one of the full axiom of choice, expecting a total choice function makes **GDC** inconsistent in its full generality.

In this section we investigate how restricting ∃**MPCF** to countable A or two-valued B impacts its strength to exactly the same extent as it restricts the strength of **GDC**. Two such preliminary results are given, but first, let's translate **GDC** in our setting:

▶ **Definition 28** (A-B-approximable). *Let A, B be types and $T : \mathcal{P}((A \times B)^*)$. For all $X : \mathcal{P}((A \times B)^*)$ define*

$$\phi(X) := \lambda u^{(A\times B)^*}.\, (u \in \lfloor\langle T\rangle\rfloor \wedge \forall a^A, \neg(\exists b^B,\ (a,b) \in u) \to \exists b^B, u@(a,b) \in X)$$

The A-B-approximation of T denoted T_{ABap} is the greatest fixed point of ϕ. We say that T is A-B-approximable if $\varepsilon \in T_{ABap}$.

▶ **Definition 29** (A-B-choice function). *Let A, B be types and $T : \mathcal{P}((A \times B)^*)$. T has an A-B-choice function if:*

$$\exists f^{A\to B}, \forall u^{(A\times B)^*}, u \subset \mathcal{G}(f) \to u \in T$$

▶ **Definition 30 (GDC).** *Let A, B be types and $T : \mathcal{P}((A \times B)^*)$, $\mathbf{GDC}_{ABT}$ is the statement: if T is A-B-approximable then T has an A-B-choice function.*

▶ **Theorem 31.** $\overline{\mathbf{GDC}_{\mathbb{N}BT}} \to_{\mathrm{cl}} \overline{\exists\mathbf{MPCF}_{\mathbb{N}BT}}$

Proof. Let B be a type and $T : \mathcal{P}((\mathbb{N} \times B)^*)$. In order to use **GDC**, T must be $\mathbb{N}$-B-approximable but the T we are given might not be. Thus, we are going to construct $T_\perp : \mathcal{P}((\mathbb{N} \times B_\perp)^*)$ that is $\mathbb{N}$-$B_\perp$-approximable and use **GDC** to obtain a function that we will prove maximal.

For all $u : \mathcal{P}((A \times B_\perp)^*)$ define $\overline{u}$ inductively:

$$\frac{}{\overline{\varepsilon} := \varepsilon} \qquad \frac{a : A \quad b : B}{\overline{u@(a,b)} := \overline{u}@(a,b)} \qquad \frac{a : A}{\overline{u@(a,\perp)} := \overline{u}}$$

By induction define $T_\perp^n : \mathcal{P}((\mathbb{N} \times B_\perp)^*)$:

- $T_\perp^0 := \lambda u^{(\mathbb{N}\times B_\perp)^*}.\, u = \varepsilon$
- Let $T_\perp^{n+1}$ be defined inductively

$$\frac{u \in T_n \quad b : B \quad \overline{u}@(n+1,b) \in T}{u@(n+1,b) \in T_\perp^{n+1}} \qquad \frac{u \in T_n \quad \forall b^B, \overline{u}@(n+1,b) \notin T}{u@(n+1,\perp) \in T_\perp^{n+1}}$$

Now define $T_\perp$ as the $\subseteq$-downward closure of the union of the $T_\perp^n$. We must show that $T_\perp$ is $\mathbb{N}$-$B_\perp$-approximable. By definition $T_\perp = \lfloor\langle T_\perp\rangle\rfloor$. Let $n : \mathbb{N}$, $v : (\mathbb{N} \times B_\perp)^*$ such that $v \in T_\perp$ and $\neg(\exists c^{B_\perp},\ (n,c) \in v)$. By definition, there exists $m : \mathbb{N}$ and $u \in T_\perp^m$ such that $v \subseteq u$. If

$n \leq m$ then there exists $c : B_\perp$ such that $(n, c) \in u$, thus $v@(n, c) \subseteq u$ and $v@(n, c) \in T_\perp$. If $n > m$ then there exists $u' \in T_\perp^n$ such that $u \subseteq u'$. It is in the proof of this statement that we need excluded middle to show that we always satisfy the hypothesis of one of the induction steps. Hence, $v \subseteq u'$ and we now repeat the same argument. $T_\perp$ satisfies ϕ and contains ε, thus we can apply $\mathbf{GDC}_{\mathbb{N}B_\perp T_\perp}$ and get $f : \mathbb{N} \to B_\perp$ a choice function.

What is left to show is that f is a maximal partial function. Let $n : \mathbb{N}$ such that $n \notin \mathbf{dom}(f)$ and let $g : \mathbb{N} \to B_\perp$ extending f with $\mathbf{dom}(g) = \mathbf{dom}(f) \cup n$. Let us write $f_{<n}$ for the list $[(0, f(0)), \ldots, (n-1, f(n-1))]$. $f_{<n} \in T_\perp^n$ and since $f_{<n+1}$ is of the form $f_{<n}@(n, \perp)$ by case analysis we deduce that $\forall b^B, \overline{f_{<n}}@(n, b) \notin T$. If $\mathcal{G}(g) \in \langle T_\perp \rangle$ then $g_{<n+1} \in T_\perp$ and $g_{<n+1} = f_{<n}@(n, g(n))$ with $g(n) : B$. $\overline{f_{<n}}@(n, g(n))$ is thus in T, contradiction. Hence, f is maximal. ◀

Let's write $\mathbf{DC}$ for the axiom of dependent choice. We have:

▶ **Corollary 32.** *Since* $\overline{\mathbf{GDC}_{\mathbb{N}BT}}$ *is equivalent to* $\mathbf{DC}$ *[2] we deduce:* $\mathbf{DC} \to_{\mathrm{cl}} \overline{\mathbf{TTL}_{(\mathbb{N}\times B)T}}$

▶ **Theorem 33.** *For A a type with decidable equality,* $\overline{\exists\mathbf{MPCF}_{A\mathbb{B}T}} \to \overline{\mathbf{GDC}_{A\mathbb{B}T}}$

Proof. Let A be a type and $T : \mathcal{P}((A \times \mathbb{B})^*)$ A-$\mathbb{B}$-approximable. Define $U := \lfloor \langle T_{A\mathbb{B}\mathrm{ap}} \rangle \rfloor$, the A-$\mathbb{B}$-approximable hypothesis guarantees that $\langle U \rangle$ is inhabited. Using $\exists\mathbf{MPCF}_{A\mathbb{B}U}$ we get $f : A \to \mathbb{B}_\perp$ a maximal partial choice function. We show that f must be total, that is that it is impossible that it takes the value $\perp$. Indeed assume $f(a) = \perp$ for some $a : A$ and consider $g : A \to \mathbb{B}_\perp$ that extends f with $g_0(a) = 0_\mathbb{B}$. We have $g \prec f$, thus $\mathcal{G}(g) \notin \langle U \rangle$. Then, there exists $u : (A \times \mathbb{B})^*$ such that $u \subset \mathcal{G}(g)$ and $u \notin U$. Using the decidability of equality in A, we can find u' such that $u = u'@(a, 0_\mathbb{B})$ where $u' \subset \mathcal{G}(f)$. Symmetrically, by considering the extension g of f obtained by setting $g(a) = 1_\mathbb{B}$, there exists $v' \subset \mathcal{G}(f)$ such that $v'@(a, 1_\mathbb{B}) \notin U$. Since $u' \star v' \subset \mathcal{G}(f)$, $u' \star v' \in U$. There must be $b : \mathbb{B}$ such that $(u' \star v')@(a, b) \in U$. But in both cases ($b = 0_\mathbb{B}$ or $1_\mathbb{B}$) there is a sublist ($u'@(a, 0_\mathbb{B})$ or $v'@(a, 1_\mathbb{B})$) that is not in U, contradiction. Hence, f is total. ◀

The following definition, taken from [2], describes how to turn a relation on A and B as a predicate over $(A \times \mathbb{B})^*$ that filters approximations.

▶ **Definition 34** (Positive alignment)**.** *Let A and B be types and R a relation on A and B. The* positive alignment $R_\top$ of R *is the predicate on $(A \times B)^*$ defined by:*

$$R_\top := \lambda u. \forall (a, b) \in u, R(a, b)$$

Positive alignments can be characterised by the following property.

▶ **Definition 35** (Downward prime)**.** *Let A and B be types. We say that $T : \mathcal{P}((A \times B)^*)$ is* downward prime *when $u \in T$ and $v \in T$ implies $u \star v \in T$. We denote by $\mathbf{D}_{AB}$ the collection of downward prime $T : \mathcal{P}((A \times B)^*)$.*

▶ **Theorem 36.** *If R is a relation on A and B, its positive alignment is downward prime. Conversely, if T is downward prime, it is the positive alignment of the relation $|T|$ defined by*

$$|T|(a, b) := [(a, b)] \in T$$

Proof. This is because $u \star v \in R_\top$, that is $\forall (a, b) \in u \star v, R(a, b)$ is equivalent to $(\forall (a, b) \in u, R(a, b)) \wedge (\forall (a, b) \in v, R(a, b))$, that is to $u \in R_\top \wedge v \in R_\top$, and, conversely, because $u \in |T|_\top$, that is $\forall (a, b) \in u, [(a, b)] \in T$, is equivalent, by induction on u, using downward primality at each step, to $u \in T$. ◀

Based on the equivalence between $\mathbf{AC}_{ABR}$ and $\mathbf{GDC}_{ABR_\top}$ in [2, Thm 7], we obtain:

▶ **Corollary 37.** *$\mathbf{GDC}_{ABT}$ for T downward prime characterises the full axiom of choice $\mathbf{AC}_{ABR}$, that is $\forall x^A, \exists y^B, R(a,b) \to \exists f^{A\to B}, \forall x^A, R(a, f(a))$.*

We now show that $\mathbf{GDC}_{ABT}$ is also equivalent to $\exists\mathbf{MPCF}_{ABT}$ for T downward prime.

▶ **Theorem 38.** *For $T : \mathcal{P}((A \times \mathbb{B})^*)$ downward prime for A with decidable equality, $\exists\mathbf{MPCF}_{ABT} \to \mathbf{GDC}_{ABT}$.*

Proof. Since T is A-B-approximable, it contains ε, so that $\langle T\rangle$ is non-empty. Thus, by $\exists\mathbf{MPCF}_{ABT}$, we get $f : A \to B_\perp$ a maximal partial choice function. We show that f must be total. Indeed, assume $a : A$ such that $f(a) = \perp$. By A-B-approximability, we can obtain a b such that $[(a,b)] \in \lfloor\langle T\rangle\rfloor$. Let's now consider the function $g : A \to B_\perp$ defined by setting $g(a') = b$ if $a = a'$ and $g(a') = f(a')$ otherwise. We have $g \prec f$, thus $\mathcal{G}(g) \notin \langle T\rangle$. But this contradicts that we can also prove that any $u \subset \mathcal{G}(g)$ is in T, that is $\mathcal{G}(g) \in \langle T\rangle$. Indeed, by decidability of equality on A, either u has an element of the form (a, b') or not. In the second case, $u \subset \mathcal{G}(f)$ and thus $u \in T$. In the first case, u has the form $u' \star (a,b') \star u''$ with $u' \in \mathcal{G}(f)$ and $u'' \in \mathcal{G}(f)$, thus $u' \in T$ and $u'' \in T$. Since $u \subset \mathcal{G}(g)$, we also have $b' = g(a) = b$. Then, by downward primality, we get $u' \star [(a,b)] \star u'' \in T$. ◀

▶ **Theorem 39.** *For $T : \mathcal{P}((A \times B)^*)$ downward prime, $\overline{\mathbf{GDC}_{AB\mathbf{D}_{AB}}} \to \overline{\exists\mathbf{MPCF}^-_{AB\mathbf{D}_{AB}}}$.*

Proof. There are two ways to embed a partial function from A to B into a total function: either restrict A to the domain of the function, or extend B into $B_\perp$, as in Theorem 31. We give a proof using the first approach.

Let A' be the subset of A such that $\exists b^B, [(a,b)] \in T$. We show coinductively that if A' is infinite, the restriction of T on A' is A'-B-approximable. First, we do have $\varepsilon \in T$ because $\langle T\rangle$ is non empty. Then, assume $u \in T$ and $a : A'$ such that $\neg(\exists b^B,\ (a,b) \in u)$ (which is possible since A' is supposed infinite). Since a is in A', there is b such that $[(a,b)] \in T$, and by downward primality, $u \star (a,b) \in T$, hence A'-B-approximable by coinduction.

Thus, there is a total function $f : A' \to B$ such that $\mathcal{G}(f) \in \langle T\rangle$, which induces a partial function f' from $A \to_{\mathrm{p}} B_\perp$. It remains to show that f' is maximal. Let $a \notin \mathbf{dom}(f)$, that is such that $\forall b^B, \neg[(a,b)] \in T$. Then, there is obviously no extension of f' on a that would be in $\langle T\rangle$.

It remains to treat the case of A' finite, which can be obtained by (artificially) reasoning on the disjoint sum of A' and $\mathbb{N}$, and setting $T[(n,p)] := (n = p)$ on $\mathbb{N}$. ◀

5 Conclusion

While Brede and the first author [2] investigated the general form of a variety of choice and bar induction principles seen as contrapositive principles, this paper initiated the investigation of a general form of maximality and well-foundedness principles equivalent to the axiom of choice. One of the surprise was that, up to logical duality, two principles such as Teichmüller-Tukey lemma and Berger's update induction were actually of the very same nature. By seeing all these principles as schemes, we could also investigate how to express Zorn's lemma and Teichmüller-Tukey lemma as mutual instances the one of the other. Finally, by starting investigating how maximality, when applied to functions, relates to totality in the presence of either a countable domain or a finite codomain, we initiated a bridge between maximality and well-foundedness principles and the general family of choice and bar induction principles from [2].

The investigation could be continued in at least five directions:

- In the articulation between **TTL** and ∃**MPCF**: assuming an alternative definition of **TTL**, say $\mathbf{TTL}^+$, where $\mathcal{P}(A)$ is represented as a characteristic function from A to $\mathbb{B}$, that is, equivalently, as a function from A to $\mathbb{1}_\perp$, one would get the following identifications:

$$\begin{array}{lcllcl} \mathbf{TTL}^+_{AT} & = & \exists\mathbf{MPCF}_{A\mathbb{1}\pi^*T} & \mathbf{TTL}^+_{(A\times B)T} & = & \exists\mathbf{MPCF}_{ABT} \\ \mathbf{TTL}_{AT} & = & \exists\mathbf{MPCF}^-_{A\mathbb{1}\pi^*T} & \mathbf{TTL}_{(A\times B)T} & = & \exists\mathbf{MPCF}^-_{ABT} \end{array}$$

- In the articulation between a sequential definition of countably-finite character and countably-open predicate, as in $\mathbf{TTL}^N_{BT}$ and $\mathbf{UI}_{BT}$, and a non-sequential definition, as in $\exists\mathbf{MPCF}_{\mathbb{N}BT}$ and $\exists\mathbf{MPCF}^-_{\mathbb{N}BT}$, similar to the connection between $\mathbf{DC}^{prod.}_{BT}$ and $\mathbf{GDC}_{\mathbb{N}BT}$ in [2].
- In the relation between $\exists\mathbf{MPCF}_{ABT}$ and $\exists\mathbf{MPCF}^-_{ABT}$ on one side and $\mathbf{GDC}_{ABT}$ on the other side, verifying that the correspondences between $\exists\mathbf{MPCF}_{\mathbb{N}BT}$ and $\mathbf{GDC}_{\mathbb{N}BT}$, and between $\exists\mathbf{MPCF}_{A\mathbb{B}T}$ and $\mathbf{GDC}_{A\mathbb{B}T}$ hold, at least classically, in both directions, the same way as they do in the case T downward prime.
- In the articulation between **TTL** and **GUI**, formulating statements dual to ∃**MPCF** and $\exists\mathbf{MPCF}^-$ and connecting them to **GBI** [2], analysing the role of classical reasoning and decidability of the equality on the domain in the correspondences.
- In the relation between **TTL**, ∃**MPCF**, $\exists\mathbf{MPCF}^-$ and other maximality principles than Zorn's lemma, also studying other well-foundedness principles than **UI**.

In particular, an advantage of ∃**MPCF** and $\exists\mathbf{MPCF}^-$ over **GDC** is that their more general form is classically equivalent to the axiom of choice while the most general form of **GDC** is inconsistent.

References

1 Ulrich Berger. A computational interpretation of open induction. In *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science*, LICS '04, page 326, USA, 2004. IEEE Computer Society.

2 Nuria Brede and Hugo Herbelin. On the logical structure of choice and bar induction principles. *36th Annual Symposium on Logic in Computer Science*, 2021.

3 Thierry Coquand. Constructive topology and combinatorics. In J. Paul Myers and Michael J. O'Donnell, editors, *Constructivity in Computer Science*, pages 159–164, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

4 Horst Herrlich. *Axiom of Choice*. Lecture Notes in Mathematics. Springer, 2006.

5 Thomas J. Jech. *The Axiom of Choice*. Dover Books on Mathematics Series. Courier corporation, 1973.

6 Herman Rubin and Jean E. Rubin. *Equivalents of the Axiom of Choice*. North-Holland Publishing Company, 1970.

A Verified Algorithm for Deciding Pattern Completeness

René Thiemann
University of Innsbruck, Austria

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

Abstract

Pattern completeness is the property that the left-hand sides of a functional program cover all cases w.r.t. pattern matching. In the context of term rewriting a related notion is quasi-reducibility, a prerequisite if one wants to perform ground confluence proofs by rewriting induction.

In order to certify such confluence proofs, we develop a novel algorithm that decides pattern completeness and that can be used to ensure quasi-reducibility. One of the advantages of the proposed algorithm is its simple structure: it is similar to that of a regular matching algorithm and, unlike an existing decision procedure for quasi-reducibility, it avoids enumerating all terms up to a given depth.

Despite the simple structure, proving the correctness of the algorithm is not immediate. Therefore we formalize the algorithm and verify its correctness using the proof assistant Isabelle/HOL. To this end, we not only verify some auxiliary algorithms, but also design an Isabelle library on sorted term rewriting. Moreover, we export the verified code in Haskell and experimentally evaluate its performance. We observe that our algorithm significantly outperforms existing algorithms, even including the pattern completeness check of the GHC Haskell compiler.

2012 ACM Subject Classification Theory of computation → Pattern matching; Theory of computation → Program verification; Theory of computation → Higher order logic

Keywords and phrases Isabelle/HOL, pattern matching, term rewriting

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.27

Supplementary Material *InteractiveResource (Website with Formalization, Experiments, Sources)*: `http://cl-informatik.uibk.ac.at/software/ceta/experiments/pat_complete/`

Funding This research was supported by the Austrian Science Fund (FWF) project I 5943.

Acknowledgements We thank Takahito Aoto and Fabian Mitterwallner for their help in conducting experiments with the tools AGCP and FORT-h, respectively; and we thank Dohan Kim for his contributions to the formalization of an auxiliary algorithm. We are grateful to all reviewers for their helpful remarks and suggestions.

1 Introduction

Consider programs written in a declarative style such as functional programs or term rewrite systems (TRSs), where evaluation is triggered by pattern matching. In many applications it is important to ensure that evaluation of a given program cannot get stuck – this property is called quasi-reducibility [8] in the context of TRSs or pattern completeness in the context of functional programming. For instance in Isabelle/HOL [14], in a function definition the patterns must cover all cases (in addition to termination), since HOL is a logic of total functions. Moreover, automated theorem proving methods that are based on rewriting induction [1, 15] require similar completeness results, e.g., for proving ground confluence.

▶ **Example 1.** Let $\mathcal{C}_{\mathbb{N}} = \{\mathsf{true} : \mathbb{B}, \mathsf{false} : \mathbb{B}, \mathsf{0} : \mathbb{N}, \mathsf{s} : \mathbb{N} \to \mathbb{N}\}$ be the set of constructors to represent the Booleans and natural numbers in Peano's notation. We consider a TRS $\mathcal{R}_{\mathbb{N}}$ that defines a function $\mathsf{even} : \mathbb{N} \to \mathbb{B}$ to compute whether a natural number is even.

$$\mathsf{even}(\mathsf{0}) \to \mathsf{true} \qquad \mathsf{even}(\mathsf{s}(\mathsf{0})) \to \mathsf{false} \qquad \mathsf{even}(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{even}(x) \tag{1}$$

This TRS is quasi-reducible, since no matter which number n we provide as argument, one of the left-hand sides (lhss) will match the term $\mathsf{even}(n)$; this fact can easily be seen by a case-analysis on whether n represents 0, 1, or some larger number. Note the importance of sorts:[1] without them, the evaluation of the (unsorted) term $\mathsf{even}(\mathsf{s}(\mathsf{true}))$ would get stuck.

Kapur et al. proved the decidability of quasi-reducibility [8]. Their decidability result does not yield a practical algorithm: it has an exponential best-case complexity, i.e., to ensure quasi-reducibility, one always has to enumerate exponentially many terms and test whether their evaluation does not get stuck. Therefore, Lazrek, Lescanne and Thiel developed a more practical approach. Their *complement algorithm* [12] is a decision procedure for pattern completeness in the left-linear case, but it might fail on TRSs that are not left-linear. Note that in the left-linear case, pattern completeness and quasi-reducibility can also be encoded into a problem about tree automata.

In this paper, we develop a novel algorithm for pattern completeness with the following key features.

- It is a decision procedure, even in the non-linear case.
- The algorithm is syntax directed and it is easy to implement.
- In our experiments it outperforms existing implementations of the complement algorithm, the approach via tree automata, and pattern completeness check by the `ghc` Haskell compiler.
- Its correctness is fully verified in Isabelle/HOL.

We are aware of two other algorithms to ensure quasi-reducibility in more complex settings, e.g., where rules may be guarded by arithmetic constraints such as "this rule is only applicable if $x > 0$" [5, 9], but both algorithms do not properly generalize the result of Kapur et al. since they are restricted to linear lhss. Bouhoula and Jacquemard [3] also designed an algorithm in a more complex setting with conditions and constraints, and a back-end that is based on constrained tree automata techniques. Since their soundness result is restricted to ground confluent systems, their algorithm is not applicable in our use case; ultimately we want to verify ground confluence proofs on methods that rely upon quasi-reducibility. Moreover, Bouhoula developed an algorithm to verify ground confluence and completeness at the same time [2], where we are not sure whether it can also be adjusted to an algorithm that just ensures completeness, e.g., for non-ground confluent systems. Furthermore, there are proof methods that ensure pattern completeness within proof assistants. These are used to ensure well-definedness of function definitions. For instance, in Isabelle/HOL there is a corresponding method `pat_completeness` [10], but as many other algorithms for pattern completeness, it is restricted to the left-linear case.

There are also algorithms to compile pattern matching [11, 16], however these have a different focus: their major aim is not to decide or to ensure completeness, but instead they generate efficient code for functional programs that are defined by pattern matching.

[1] A *sort* in the TRS context is the same as a *type* when speaking about functional programs. Since most of this paper is written using TRS notation, we speak of sorts instead of types in the rest of the paper.

The paper is organized as follows: In Section 2 on preliminaries we introduce notions and notations, and recall the core concepts of pattern completeness and quasi-reducibility. Then in Section 3 we present the first part of our novel algorithm that covers the linear case. The algorithm is then extended to handle the general case in Section 4. Afterwards we present details on the Isabelle formalization and on the implementation in Section 5. The experimental results are provided in Section 6 before we conclude in Section 7.

The formalization, the executable code and details on the experiments are available at:

`http://cl-informatik.uibk.ac.at/software/ceta/experiments/pat_complete/`

2 Preliminaries

We fix a set $\mathcal{S}$ of sorts. A sorted set A is a set where each element a is associated with a sort $\iota \in \mathcal{S}$, written $a : \iota \in A$. A sorted signature $\mathcal{F}$ is a set of function symbols f, each associated with a nonempty sequence of sorts $\iota_1, \ldots, \iota_n, \iota_0 \in \mathcal{S}$, written $f : \iota_1 \times \cdots \times \iota_n \to \iota_0 \in \mathcal{F}$. Given a sorted signature $\mathcal{F}$ and a sorted set $\mathcal{V}$ of variables, the sorted set $\mathcal{T}(\mathcal{F}, \mathcal{V})$ of terms is defined as follows: $x : \iota \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ if $x : \iota \in \mathcal{V}$; and $f(t_1, \ldots, t_n) : \iota_0 \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ if $f : \iota_1 \times \cdots \times \iota_n \to \iota_0 \in \mathcal{F}$ and $t_1 : \iota_1, \ldots, t_n : \iota_n \in \mathcal{T}(\mathcal{F}, \mathcal{V})$. We denote the set of variables occurring in t by $\mathsf{Var}(t)$. By $\mathcal{T}(\mathcal{F})$ we denote the sorted set of ground terms, i.e., terms that do not contain variables. A term is linear, if it does not contain any variable more than once. A sorted map f from a sorted set A to a sorted set B, written $f : A \to B$, is a map such that $f(a) : \iota \in B$ whenever $a : \iota \in A$. A substitution is a sorted map $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}, \mathcal{V})$ for another sorted set $\mathcal{X}$ of variables,[2] and the instance is the term $t\sigma \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ obtained from $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ by replacing all x by $\sigma(x)$. We write $\sigma\delta$ for the composition of two substitutions σ and δ, and $[x \mapsto t]$ is the substitution which substitutes x by t and $y[x \mapsto t] = y$ for all $x \neq y$. A term $\ell : \iota \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ matches a term $t : \iota \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ if there exists a substitution $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}, \mathcal{V})$ such that $\ell\sigma = t$.

We consider programs that consist of a set of rules $\ell \to r$ and evaluation is defined by replacing instances of left-hand sides (lhss) $\ell\sigma$ by instances of right-hand sides $r\sigma$. For instance a program might be a TRS, or some other first-order functional programming language that uses pattern matching. We assume that there is a fixed finite signature $\mathcal{F} = \mathcal{C} \uplus \mathcal{D}$, where $\mathcal{C}$ contains constructor symbols and $\mathcal{D}$ contains defined symbols. Hence, input values to a function are represented by constructor ground terms. We assume that there exists a constructor ground term $t : \iota \in \mathcal{T}(\mathcal{C})$ for each sort $\iota \in \mathcal{S}$. A sort ι is finite if $\{t \mid t : \iota \in \mathcal{T}(\mathcal{C})\}$ is a finite set, otherwise ι is infinite. We say a term $f(t_1, \ldots, t_n)$ is *basic* if $f : \iota_1 \times \cdots \times \iota_n \to \iota_0 \in \mathcal{D}$ and $t_1 : \iota_1, \ldots, t_n : \iota_n \in \mathcal{T}(\mathcal{C}, \mathcal{V})$, and denote the set of basic terms by $\mathcal{B}(\mathcal{C}, \mathcal{D}, \mathcal{V})$.

We are now ready to formally describe that evaluation of a program cannot get stuck.

▶ **Definition 2** (Pattern Completeness of Programs). *A program with lhss L is pattern complete, if every basic ground term $t \in \mathcal{B}(\mathcal{C}, \mathcal{D}, \emptyset)$ is matched by some $\ell \in L$.*

Pattern completeness is an instance of the *cover problem*, a notion that appears in the context of both term rewriting and functional programming [4, 11, 18]: a set L of terms *covers* a set T of terms if every constructor ground instance of a term in T is matched by some term in L. Clearly, pattern completeness of a program with lhss L is exactly the

[2] On paper it is not essential to distinguish the sets of variables, while it is convenient in the formalization that we can use variables different from those used to represent programs.

property that L covers $T := \{f(x_1, \ldots, x_n) \mid f : \iota_1 \times \cdots \times \iota_n \to \iota_0 \in \mathcal{D}\}$ where $x_1, \ldots, x_n$ is a sequence of distinct variables; and the complement algorithm of Lazrek et al. [12] can be used to decide whether L covers T if all terms in L are linear.

An alternative notion to pattern completeness is quasi-reducibility [8], where the difference is that matching can happen for an arbitrary subterm.

▶ **Definition 3** (Quasi-Reducibility of Programs). *A program with lhss L is quasi-reducible, if every basic ground term $t \in \mathcal{B}(\mathcal{C}, \mathcal{D}, \emptyset)$ contains a subterm that is matched by some $\ell \in L$.*

Pattern completeness implies quasi-reducibility since t is a subterm of t, and the two notions coincide if the root symbols of all lhss are in $\mathcal{D}$ – as in the functional programming setting or in Example 1. Example 4 illustrates the difference between the two notions.

▶ **Example 4.** Consider $\mathcal{C}_\mathbb{Z} = \{\mathsf{true} : \mathbb{B}, \mathsf{false} : \mathbb{B}, \mathsf{0} : \mathbb{Z}, \mathsf{s} : \mathbb{Z} \to \mathbb{Z}, \mathsf{p} : \mathbb{Z} \to \mathbb{Z}\}$ to represent the Booleans and integers in a successor–predecessor notation, e.g., $\mathsf{p}(\mathsf{0})$ represents -1. Now we consider a TRS $\mathcal{R}_\mathbb{Z}$ that defines a function to compute whether an integer is even, i.e., $\mathcal{D} = \{\mathsf{even} : \mathbb{Z} \to \mathbb{B}\}$. It consists of all rules of $\mathcal{R}_\mathbb{N}$ in Example 1 and the following rules.

$$\mathsf{even}(\mathsf{p}(\mathsf{0})) \to \mathsf{false} \qquad \mathsf{even}(\mathsf{p}(\mathsf{p}(x))) \to \mathsf{even}(x) \tag{2}$$

$$\mathsf{s}(\mathsf{p}(x)) \to x \qquad \mathsf{p}(\mathsf{s}(x)) \to x \tag{3}$$

This TRS is quasi-reducible since every term $\mathsf{even}(n)$ with $n : \mathbb{Z} \in \mathcal{T}(\mathcal{C}_\mathbb{Z})$ has a subterm that is matched by some lhs: If n contains both s and p then one of the rules (3) is applicable. Otherwise n is of the form $\mathsf{s}^i(\mathsf{0})$ or $\mathsf{p}^i(\mathsf{0})$ and then rules (1) or (2) will be applicable.

The TRS is not pattern complete since $\mathsf{even}(\mathsf{s}(\mathsf{p}(\mathsf{0})))$ is not matched by any lhs.

3 Pattern Completeness – The Linear Case

Before we design the new decision procedure for pattern completeness we first reformulate and generalize this notion, leading to matching problems and pattern problems.

▶ **Definition 5** (Matching Problem and Pattern Problem). *A* matching problem *is a finite set $mp = \{(t_1, \ell_1), \ldots, (t_n, \ell_n)\} \subseteq \mathcal{T}(\mathcal{F}, \mathcal{V}) \times \mathcal{T}(\mathcal{F}, \mathcal{X})$ of pairs of terms. A* pattern problem *is a finite set $pp = \{mp_1, \ldots, mp_k\}$ of matching problems.*

A matching problem mp is complete *w.r.t. a constructor ground substitution $\sigma : \mathcal{V} \to \mathcal{T}(\mathcal{C})$ if there is some substitution $\gamma : \mathcal{X} \to \mathcal{T}(\mathcal{F})$ such that $t\sigma = \ell\gamma$ for all $(t, \ell) \in mp$. A pattern problem pp is complete if for each constructor ground substitution σ there is some $mp \in pp$ such that mp is complete w.r.t. σ. A set P of pattern problems is complete if each $pp \in P$ is complete.*

When expanding the definition of completeness of a set of pattern problems P we obtain an alternative definition, which reveals that there are two quantifier alternations.

$$P \text{ is complete} \;\text{ iff }\; \forall pp \in P.\, \forall \sigma : \mathcal{V} \to \mathcal{T}(\mathcal{C}).\, \exists mp \in pp.\, \exists \gamma : \mathcal{X} \to \mathcal{T}(\mathcal{F}).\, \forall (t, \ell) \in mp.\, t\sigma = \ell\gamma$$

Pattern problems are quite generic and can express several properties. For instance, L covers T iff the set of pattern problem $P = \{\{\{(t, \ell)\} \mid \ell \in L\} \mid t \in T\}$ is complete. Similarly, Aoto and Toyama's notion of strong quasi-reducibility [1] can also be encoded as a pattern problem: $pp = \{\{(t, \ell)\} \mid t \in \{x_1, \ldots, x_n, f(x_1, \ldots, x_n)\},\ \ell \in L\}$ expresses that one tries to find a match at the root ($t = f(x_1, \ldots, x_n)$) or a match for a direct subterm ($t = x_i$). Finally, the question of whether a program with lhss L and defined symbols $\mathcal{D}$ is pattern complete w.r.t. Definition 2 is expressible as the completeness of the set of pattern problems $P = \{\{\{(f(x_1, \ldots, x_n), \ell)\} \mid \ell \in L\} \mid f : \iota_1 \times \cdots \times \iota_n \to \iota_0 \in \mathcal{D}\}$.

The following inference rules describe a decision procedure to determine the completeness of *linear* pattern problems. A matching problem $\{(t_1, \ell_1), \ldots, (t_n, \ell_n)\}$ is linear if each ℓ_i is linear and $\mathsf{Var}(\ell_i) \cap \mathsf{Var}(\ell_j) = \emptyset$ for $i \neq j$. We say a pattern problem is *linear* if all its matching problems are linear.

In the inference rules we make use of a special matching problem $\bot_{mp}$ that represents an incomplete matching problem. Similarly, we define $\top_{pp}$ as a new pattern problem that is always complete. Finally, $\bot_P$ represents a new incomplete set of pattern problems.

▶ **Definition 6** (Inference Rules for Linear Pattern Problems). *We define $\to$ as the set of the following simplification rules for matching problems.*

$$\{(f(t_1, \ldots, t_n), f(\ell_1, \ldots, \ell_n))\} \uplus mp \to \{(t_1, \ell_1), \ldots, (t_n, \ell_n)\} \cup mp \qquad \text{(decompose)}$$

$$\{(t, x)\} \uplus mp \to mp \qquad \text{if } \forall (t', \ell) \in mp.\ x \notin \mathsf{Var}(\ell) \qquad \text{(match)}$$

$$\{(f(\ldots), g(\ldots))\} \uplus mp \to \bot_{mp} \qquad \text{if } f \neq g \qquad \text{(clash)}$$

On top of this we define the set $\Rightarrow$ of simplification rules for pattern problems as follows:

$$\{mp\} \uplus pp \Rightarrow \{mp'\} \cup pp \qquad \text{if } mp \to mp' \qquad \text{(simp-mp)}$$

$$\{\bot_{mp}\} \uplus pp \Rightarrow pp \qquad \text{(remove-mp)}$$

$$\{\emptyset\} \uplus pp \Rightarrow \top_{pp} \qquad \text{(success)}$$

Finally we provide rules $\Rrightarrow$ for modifying sets of pattern problems.

$$\{pp\} \uplus P \Rrightarrow \{pp'\} \cup P \qquad \text{if } pp \Rightarrow pp' \qquad \text{(simp-pp)}$$

$$\{\emptyset\} \uplus P \Rrightarrow \bot_P \qquad \text{(failure)}$$

$$\{\top_{pp}\} \uplus P \Rrightarrow P \qquad \text{(remove-pp)}$$

$$\{pp\} \uplus P \Rrightarrow \mathsf{Inst}(pp, x) \cup P \qquad \text{if } mp \in pp \text{ and } (x, f(\ldots)) \in mp \qquad \text{(instantiate)}$$

Here, for a pattern problem pp and a variable $x : \iota_0 \in \mathcal{V}$, the pattern problem set $\mathsf{Inst}(pp, x)$ consists of a pattern problem $pp\sigma_{x,c} = \{\{(t\sigma_{x,c}, \ell) \mid (t, \ell) \in mp\} \mid mp \in pp\}$ for each $c : \iota_1 \times \cdots \times \iota_n \to \iota_0 \in \mathcal{C}$, where $\sigma_{x,c} = [x \mapsto c(x_1, \ldots, x_n)]$ for distinct fresh variables $x_1 : \iota_1, \ldots, x_n : \iota_n \in \mathcal{V}$.

Clearly, (decompose), (match) and (clash) correspond to a standard matching algorithm. Most of the other rules correspond to the universal and existential quantification that is done in the definition of completeness. The only exception is (instantiate). Here a matching algorithm would detect a failure since a variable x is never matched by a non-variable term $f(\ldots)$. However, since the x in our setting just represents an arbitrary constructor ground term, we need to do case analysis on the outermost constructor. This is done by replacing $x : \iota_0 \in \mathcal{V}$ by all possible constructor terms of shape $c(x_1, \ldots, x_n)$ for all $c : \iota_1 \times \cdots \times \iota_n \to \iota_0 \in \mathcal{C}$.

The following theorem states that $\Rrightarrow$ can be used to decide completeness of linear pattern problems. Here, $\Rrightarrow^!$ is defined as reduction to normal form, i.e., $P \Rrightarrow^! P'$ iff $P \Rrightarrow^* P' \wedge \nexists P''.\ P' \Rrightarrow P''$.

▶ **Theorem 7** (Decision Procedure for Completeness of Linear Pattern Problems).

- *$\Rrightarrow$ is terminating.*
- *If $P \Rrightarrow P'$ then P is complete iff P' is complete.*
- *If P is linear and $P \Rrightarrow P'$, then P' is linear.*
- *If P is linear and $P \Rrightarrow^! P'$ then $P' \in \{\emptyset, \bot_P\}$.*
- *If P is linear, then P is complete iff $P \Rrightarrow^! \emptyset$.*

Proof. The property that normal forms will be either $\emptyset$ or $\bot_P$ follows by an easy analysis of the rules. In particular (clash), (decompose), (match), and (instantiate) cover all cases of a pair (t, ℓ) of a linear matching problem, i.e., whether these terms are variables or function applications, and whether the root symbol of both terms is identical or not. Note that the condition in (match) is always satisfied for linear matching problems.

Preserving completeness is rather obvious for all rules except (instantiate), which was already explained in the paragraph directly after Definition 6.

The most interesting aspect is termination. To prove it, we first define a measure of difference $|\ell - t|$ for a term pair (t, ℓ) of some matching problem:

- $|\ell - x|$ is the number of function symbols in ℓ,
- $|f(\ell_1, \dots, \ell_n) - f(t_1, \dots, t_n)| = \sum_{i=1}^{n} |\ell_i - t_i|$, and
- $|\ell - t| = 0$ in all other cases.

We lift this measure to pattern problems by $|pp|_{\text{diff}} = \sum_{mp \in pp, (t,\ell) \in mp} |\ell - t|$. Finally, we define $\succ$ as a relation on sets of pattern problems via the multiset extension $>^{mul}$ of $>$ by $P \succ P'$ iff $\{|pp|_{\text{diff}} \mid pp \in P\} >^{mul} \{|pp|_{\text{diff}} \mid pp \in P'\}$. This relation is strongly normalizing and has the nice property that each $\Rrightarrow$-step weakly decreases and the (instantiate) rule strictly decreases w.r.t. $\succ$. Hence, (instantiate) cannot be applied infinitely often. That the remaining rules terminate does not need a complicated measure: their application decreases the number of function symbols or the cardinalities of the sets. ◀

So, completeness of linear pattern problems is decidable. Let us illustrate the algorithm on a previous example.

▶ **Example 8.** The algorithm validates that $\mathcal{R}_{\mathbb{N}}$ in Example 1 is pattern complete.

$$
\begin{aligned}
P &= \{\, \{\{(\mathsf{even}(y), \mathsf{even}(\mathsf{0}))\}, \{(\mathsf{even}(y), \mathsf{even}(\mathsf{s}(\mathsf{0})))\}, \{(\mathsf{even}(y), \mathsf{even}(\mathsf{s}(\mathsf{s}(x))))\}\} \,\} \\
&\Rrightarrow^* \{\, \{\{(y, \mathsf{0})\}, \{(y, \mathsf{s}(\mathsf{0}))\}, \{(y, \mathsf{s}(\mathsf{s}(x)))\}\} \,\} \\
&\Rrightarrow \{\, \{\{(\mathsf{0}, \mathsf{0})\}, \{(\mathsf{0}, \mathsf{s}(\mathsf{0}))\}, \{(\mathsf{0}, \mathsf{s}(\mathsf{s}(x)))\}\},\ \{\{(\mathsf{s}(z), \mathsf{0})\}, \{(\mathsf{s}(z), \mathsf{s}(\mathsf{0}))\}, \{(\mathsf{s}(z), \mathsf{s}(\mathsf{s}(x)))\}\} \,\} \\
&\Rrightarrow^* \{\, \{\emptyset, \bot_{mp}, \bot_{mp}\},\ \{\bot_{mp}, \{(z, \mathsf{0})\}, \{(z, \mathsf{s}(x))\}\} \,\} \\
&\Rrightarrow^* \{\, \{\{(z, \mathsf{0})\}, \{(z, \mathsf{s}(x))\}\} \,\} \\
&\Rrightarrow \{\, \{\{(\mathsf{0}, \mathsf{0})\}, \{(\mathsf{0}, \mathsf{s}(x))\}\},\ \{\{(\mathsf{s}(y), \mathsf{0})\}, \{(\mathsf{s}(y), \mathsf{s}(x))\}\} \,\} \\
&\Rrightarrow^* \{\, \{\emptyset, \bot_{mp}\},\ \{\bot_{mp}, \emptyset\} \,\} \\
&\Rrightarrow^* \emptyset
\end{aligned}
$$

Regarding the complexity of the algorithm, one can prove an exponential upper bound on the number of $\Rrightarrow$-steps. Note that deciding quasi-reducibility is co-NP complete [7], and this result carries over to pattern completeness: the restriction to just search for matches at the root position in pattern completeness does not cause a lower complexity class. So unless P = NP, the exponential upper bound cannot be improved.

We briefly illustrate the idea of why pattern completeness is co-NP hard.

▶ **Theorem 9.** *Deciding pattern completeness is co-NP hard for both TRSs and pattern problems, even in the linear case.*

Proof. We perform a reduction from the Boolean satisfiability problem for conjunctive normal forms (CNFs). So assume φ is a CNF that contains n different Boolean variables $x_1, \dots, x_n$ and consists of m clauses $c_1, \dots, c_m$. W.l.o.g. we assume that no clause c_i contains conflicting literals, i.e., x_j and $\neg x_j$ for the same j, because such clauses are trivially valid and can therefore be removed from the set of clauses in polynomial time.

We translate φ into the following left-linear TRS. We use a signature that contains the two Booleans true and false as constructors, and there is one defined symbol $\mathsf{f} : \mathbb{B} \times \ldots \times \mathbb{B} \to \mathbb{B}$ of arity n. We then define the TRS $\mathcal{R}_\varphi$ over this signature. It contains m rules where the ith rule has the form $\ell_i := \mathsf{f}(t_{i1}, \ldots, t_{in}) \to \mathsf{true}$ and t_{ij} is defined as false if $x_j \in c_i$, true if $\neg x_j \in c_i$, and x_j, otherwise. For instance, if $n = 5$ and $c_2 = \{\neg x_1, x_3, x_4\}$ then $\ell_2 = \mathsf{f}(\mathsf{true}, x_2, \mathsf{false}, \mathsf{false}, x_5)$. It is clear that this translation is computable in time proportional to $n \times m$.

For every variable assignment $\alpha : \{x_1, \ldots, x_n\} \to \mathbb{B}$, let f_α be the term $\mathsf{f}(\alpha(x_1), \ldots, \alpha(x_n))$. Consequently, for every α and clause c_i we see that $\alpha \models c_i$ iff f_α is not matched by ℓ_i. Therefore, φ is unsatisfiable iff $\mathcal{R}_\varphi$ is pattern complete.

Of course, instead of using the TRS $\mathcal{R}_\varphi$, one can equivalently use the linear pattern problem $pp_\varphi := \{\{(f(x_1, \ldots, x_n), \ell_1)\}, \ldots, \{(f(x_1, \ldots, x_n), \ell_m)\}\}$ and derive that φ is unsatisfiable iff pp_φ is complete. ◀

4 Pattern Completeness – The General Case

In the non-linear case $\Rrightarrow$ might get stuck, e.g., if there is a matching problem $\{(t, x), (t', x)\}$ for $t \neq t'$. To treat such cases we have to add further simplification rules. In order to do so without breaking termination, we need to distinguish between finite and infinite sorts. To illustrate the problem, consider a TRS to determine the majority of three values: $\{f(x, x, y) \to x, f(x, y, x) \to x, f(y, x, x) \to x\}$. If x is a variable of a finite sort, then the lhss can be pattern complete: if the sort allows at most two different values, such as the Booleans, then the lhss cover all cases. If the sort has at least three values a, b, c, then no lhs matches $f(a, b, c)$. So, we want to allow instantiating variables to judge pattern completeness, but we cannot allow instantiating variables of infinite sorts, since otherwise the resulting inference rules would no longer be terminating.

As final preparation for the new inference rules we define (the only two) reasons on why two terms differ.

▶ **Definition 10.** *We say that terms t and t'* clash *if $t|_p = f(\ldots) \neq g(\ldots) = t'|_p$ with $f \neq g$ for some shared position p of t and t'. The terms t and t'* differ in variable x *if $t|_p \neq t'|_p$ and $x \in \{t|_p, t'|_p\}$ for some shared position p.*

▶ **Definition 11** (Inference Rules for General Pattern Problems)**.** *We take all rules of Definition 6 and add the following ones.*

$$\{(t, x), (t', x)\} \uplus mp \to \bot_{mp} \qquad \text{if } t \text{ and } t' \text{ clash} \qquad \text{(clash')}$$

$$\{pp\} \uplus P \Rrightarrow \mathsf{Inst}(pp, x) \cup P \qquad \text{(instantiate')}$$

if $mp \in pp$, $\{(t, y), (t', y)\} \subseteq mp$, t and t' differ in variable $x : \iota \in \mathcal{V}$, and ι is finite

$$\{pp\} \uplus P \Rrightarrow \bot_P \qquad \text{if for each } mp \in pp \text{ there are } \{(t, y), (t', y)\} \subseteq mp \qquad \text{(failure')}$$

such that t and t' differ in variable $x : \iota \in \mathcal{V}$ and ι is infinite

Indeed, with these new rules, $\Rrightarrow$ cannot get stuck even for non-linear inputs.

We first remark that there is a different flavor of problems with non-linear matching problems of the form $\{(t, x), (t', x)\}$. A clash of t and t' can always be resolved locally by (clash'). If there is a difference in a finite-sort variable, this can also be handled locally by (instantiate'). However, differences in infinite-sort variables can only be applied via (failure') if indeed all matching problems show such a difference. Note that it is unsound to turn (failure') into a local rule for matching problems, i.e., if we would make (failure') similar to (clash').

▶ **Example 12.** Consider lhss $\{f(x,x), f(\mathsf{s}(x),y), f(x,\mathsf{s}(y))\}$, where $f : \mathbb{N}\times\mathbb{N}\to\mathbb{N}$ is defined and $\mathsf{0} : \mathbb{N}$ and $\mathsf{s} : \mathbb{N}\to\mathbb{N}$ are the constructors. We can start to apply the rules as follows:

$$\begin{aligned}&\{\{(f(x_1,x_2), f(x,x))\}, \{(f(x_1,x_2), f(\mathsf{s}(x),y))\}, \{(f(x_1,x_2), f(x,\mathsf{s}(y)))\}\}\\ \Rrightarrow^*&\{\{(x_1,x),(x_2,x)\}, \{(x_1,\mathsf{s}(x))\}, \{(x_2,\mathsf{s}(y))\}\} =: pp\end{aligned}$$

If at this point we would remove the matching problem $\{(x_1,x),(x_2,x)\}$ from pp because of a difference in variable x_1, then we would switch from the complete pattern problem pp into the incomplete pattern problem $pp \setminus \{\{(x_1,x),(x_2,x)\}\}$: the constructor ground substitution σ where $\sigma(x_1) = \sigma(x_2) = \mathsf{0}$ is covered by pp, but not by $pp \setminus \{\{(x_1,x),(x_2,x)\}\}$.

We state a similar theorem to the linear case, though its proof is much more evolved.

▶ **Theorem 13** (Decision Procedure for Completeness of Pattern Problems)**.** *Consider $\Rrightarrow$ of Definition 11.*

- *$\Rrightarrow$ is terminating.*
- *If $P \Rrightarrow^! P'$ then $P' \in \{\emptyset, \bot_P\}$.*
- *If $P \Rrightarrow P'$ then P is complete iff P' is complete.*
- *P is complete iff $P \Rrightarrow^! \emptyset$.*

Proof. Not getting stuck is the easiest of these properties and we leave it as an exercise to the interested reader to show that the only possible normal forms are $\emptyset$ and $\bot_P$.

Termination is more complicated. We reuse the difference measure $|\cdot|_{\text{diff}}$ from the linear case, and combine it with three other measures:

- $|pp|_{\text{symbols}}$ counts the number of all function symbols within pp.
- $|pp|_{\text{set-size}}$ sums up the cardinalities of all sets, while ignoring the term structure.
- $|pp|_{\text{fin-vars}}$ is used to measure the size of variables of finite sort. To be more precise, for a variable x having some finite sort ι, we first define $|x|$ as the maximal term size of a constructor ground term of sort ι. On top of this auxiliary measure we define

$$|pp|_{\text{fin-vars}} = \sum_{x\in\{x.\ mp\in pp,(t,\ell)\in mp,x:\iota\in\mathsf{Var}(t),\iota\text{ is finite}\}} |x|$$

  In other words, for $|\cdot|_{\text{fin-vars}}$ we collect all variables of finite sort and sum up the maximal term sizes.

The order $\succ$ on pattern problems is then defined as the lexicographic comparison of the quadruples $(|pp|_{\text{diff}}, |pp|_{\text{fin-vars}}, |pp|_{\text{symbols}}, |pp|_{\text{set-size}})$ that are obtained from a pattern problem pp. Multisets of pattern problems P are compared by $\succ^{mul}$.

All rules on sets of pattern problems give rise to a decrease w.r.t. $\succ^{mul}$, as it is indicated in the following matrix:

	$\lvert\cdot\rvert_{\text{diff}}$	$\lvert\cdot\rvert_{\text{fin-vars}}$	$\lvert\cdot\rvert_{\text{symbols}}$	$\lvert\cdot\rvert_{\text{set-size}}$
(instantiate)	$>$	$-$	$-$	$-$
(instantiate')	$\geq$	$>$	$-$	$-$
(simp-pp)	$\geq$	$\geq$	$\geq$ or $>$	$>$ or $-$
(remove-pp)	$\geq$	$\geq$	$\geq$	$>$

where the decrease in (simp-pp) depends on underlying applied rules. For instance, (decompose) leads to a decrease w.r.t. $|\cdot|_{\text{symbols}}$, but might increase $|\cdot|_{\text{set-size}}$, whereas (remove-mp) does not change $|\cdot|_{\text{symbols}}$, but gives a decrease w.r.t. $|\cdot|_{\text{set-size}}$. The reason for the decrease of (instantiate') is that whenever we instantiate some variable x of finite sort ι by $c(x_1,\ldots,x_n)$, then $|x| \geq 1 + |x_1| + \ldots + |x_n| > |x_1| + \ldots + |x_n|$, and hence the $|\cdot|_{\text{fin-vars}}$-measure strictly decreases.

Partial correctness is the most challenging part, where the difficulty is the new rule (failure'). We prove that pp is not complete, whenever (failure') is applied for pp. To this end, we build a constructor ground substitution σ such that $pp\sigma$ contains a clash.

In detail: We use an invariant φ on pattern problems defined as follows: $\varphi(pp)$ iff for each $mp \in pp$ there are y, t, t' such that $\{(t, y), (t', y)\} \subseteq mp$, t and t' contain no variable of finite sort, and moreover these terms clash or differ in a variable of infinite sort. Suppose that (failure') is applicable for a pattern problem pp. To initially satisfy φ, we apply an initial partial ground substitution σ_0 on pp, where all variables of finite sort are instantiated by some constructor ground term of that sort. Since (failure') is applicable, we know that all matching problems in $pp\sigma_0$ still have a difference in a variable of infinite sort, since none of these variables gets instantiated by σ_0.

Having that $pp\sigma_0$ satisfies the invariant φ, we now iteratively remove all variables of infinite sort that cause a difference. To be more precise, whenever some t and t' differ in a variable x in $pp\sigma_i$, we build a constructor ground term u that is larger than any of the terms in $pp\sigma_i$. Note that such u exists, since x has an infinite sort. We let σ_{i+1} to be $\sigma_i[x \mapsto u]$. This process will terminate in finitely many steps, since the set of variables in $pp\sigma_{i+1}$ is a strict subset of that of $pp\sigma_i$.

At the end of the process, we obtain a partial constructor ground substitution σ_n that instantiates all variables that contribute to differences into ground terms. Assuming that φ is maintained in the process, it is easy to see that the final pattern problem is not complete: all matching problems of the final pattern problem contain a clash of some terms t and t', i.e., we can extend σ_n to any constructor ground substitution σ that instantiates the remaining variables.

Finally we prove that the step from $pp\sigma_i$ to $pp\sigma_{i+1}$ indeed preserves φ. To this end, pick any terms t and t' of matching problem $mp \in pp\sigma_i$ that cause the conflict within the invariant. We show by case analysis that also $t[x \mapsto u]$ and $t'[x \mapsto u]$ result in a conflict.

- if t and t' clash at position p, then there also is a clash of $t[x \mapsto u]$ and $t'[x \mapsto u]$ at the same position p;
- if $t|_p = y \neq t'|_p$ for some $y \neq x$, then $t|_p[x \mapsto u] = y \neq t'|_p[x \mapsto u]$ shows that again there is a difference in y;
- if $t|_p = x \neq t'|_p$ and $t'|_p$ contains a variable $y \neq x$, then $t|_p[x \mapsto u] = u$ is a ground term and thus must have a difference with $t'|_p[x \mapsto u]$ in the variable y;
- if $t|_p = x \neq t'|_p$ and $t'|_p$ is a ground term, then $t|_p[x \mapsto u] = u \neq t'|_p = t'|_p[x \mapsto u]$, since u is strictly larger than any term of $pp\sigma_i$, and therefore there must be a clash; and finally
- if $t|_p = x \neq t'|_p$ and $\mathsf{Var}(t'|_p) = \{x\}$, then $t|_p[x \mapsto u] = u$ and $t'|_p[x \mapsto u]$ contains u as a strict subterm. Hence $t|_p[x \mapsto u]$ and $t'|_p[x \mapsto u]$ are two different ground terms which must contain a clash. ◀

Note that $\Rrightarrow$ can be modified to support counter-example generation, i.e., instead of $\bot_P$ one returns a constructor substitution that violates the completeness of a set of pattern problems. To achieve this, for each pattern problem we additionally store a constructor substitution δ. This substitution is initially empty, i.e., nothing is instantiated, and δ will keep track of the instantiations that are performed during the algorithm. The inference rules of $\Rrightarrow$ are modified as follows

$$\{(pp, \delta)\} \uplus P \Rrightarrow \{(pp', \delta)\} \cup P \quad \text{if } pp \Rightarrow pp' \qquad \text{(simp-pp)}$$

$$\{(\top_{pp}, \delta)\} \uplus P \Rrightarrow P \qquad \text{(remove-pp)}$$

$$\{(\emptyset, \delta)\} \uplus P \Rrightarrow \delta \qquad \text{(failure)}$$

$$\{(pp, \delta)\} \uplus P \Rrightarrow \{(pp\sigma_{x,c}, \delta\sigma_{x,c}) \mid c : \cdots \in \mathcal{C}\} \cup P \quad \text{if } \ldots \qquad \text{(instantiate(')')}$$

$$\{(pp, \delta)\} \uplus P \Rrightarrow \delta\sigma \quad \text{if } \ldots \qquad \text{(failure')}$$

where the dots refer to the conditions of the original rules, and in (failure') the substitution σ is defined as the constructor ground substitution that is described in the partial correctness proof of the (failure') rule in Theorem 13.

5 Formalization and Implementation

In the formalization we first describe our development on sorted term rewriting (Section 5.1, 1706 lines of Isabelle), which is essential to develop the formalization of the decision procedure of Section 4 in Isabelle. The latter uses three different layers of abstraction. We start with an abstract set based formalization (Section 5.2, 1125 lines), then refine it to a version based on multisets (Section 5.3, 828 lines), and finally arrive at an executable version that uses lists to represent matching and pattern problems (Section 5.4, 1322 lines).

5.1 Sorted Term Rewriting

Here we present our formalization of sorted term rewriting. First, we reuse the datatype for terms from the AFP entry First-Order Terms [17].

datatype $('f,'v)\ term = Var\ 'v \mid Fun\ 'f\ (('f,'v)\ term\ list)$

This datatype collects all unsorted, variadic terms. Next we introduce a sorted set over the datatype $('f,'v)\ term$. We characterize a sorted set as a partial map that assigns an element a sort. It is partial, in the sense that unsorted elements are not assigned a sort. Partial maps are readily supported in Isabelle/HOL as types of form $'a \rightharpoonup 's$, which is a synonym of $'a \to 's\ option$. We just introduce the notation "$a : \iota\ in\ A$" to mean that a sorted set $A :: 'a \rightharpoonup 's$ assigns its element $a :: 'a$ a sort $\iota :: 's$.

definition ... **where** $a : \iota\ in\ A \equiv A\ a = Some\ \iota$

Hereafter, we often omit Isabelle specifications for introducing notations by "...".

We formalize sorted signatures also as partial maps:

type_synonym $('f,'s)\ ssig = 'f \times 's\ list \rightharpoonup 's$

and introduce the following notation:

definition ... **where** $f : \iota s \to \iota_0\ in\ F \equiv F\ (f,\iota s) = Some\ \iota_0$

Given a sorted signature $F :: ('f,'s)\ ssig$ and a sorted set $V :: 'v \rightharpoonup 's$ of variables, we define the sorted set $\mathcal{T}(F,V) :: ('f,'v)\ term \rightharpoonup 's$ of terms so that

$$Var\ v : \iota\ in\ \mathcal{T}(F,V) \iff v : \iota\ in\ V$$
$$Fun\ f\ ts : \iota_0\ in\ \mathcal{T}(F,V) \iff (\exists \iota s.\ f : \iota s \to \iota_0\ in\ F \wedge ts :_l \iota s\ in\ \mathcal{T}(F,V))$$

Here, $as :_l \iota s\ in\ A$ denotes that the lists as and ιs have the same length and the ith element of as has the ith sort of ιs. We also introduce the notation $\emptyset$ for $Map.empty$, the partial map such that $\emptyset\ a = None$ for any a. Then $\mathcal{T}(F,\emptyset)$ represents the sorted set of ground terms. Given two signatures C and D, the set of basic terms is formalized as follows:

definition ... **where**
$\mathcal{B}(C,D,V) = \{Fun\ f\ ts \mid f\ \iota s\ \iota_0\ ts\ .\ f : \iota s \to \iota_0\ in\ D \wedge ts :_l \iota s\ in\ \mathcal{T}(C,V)\}$

A *sorted map* from a sorted set A to a sorted set B, written $f :_s A \to B$ in Isabelle, is a map f such that $a : \iota \; in \; A \implies f\ a : \iota \; in \; B$. In particular, sorted maps of form $\sigma :_s X \to \mathcal{T}(F,V)$ are the sorted substitutions. The application of a substitution σ on a term t is already defined as $t \cdot \sigma$ in the library in the unsorted setting. We additionally provide facts such as

lemma *subst_hastype*: $\sigma :_s X \to \mathcal{T}(F,V) \implies t : \iota \; in \; \mathcal{T}(F,X) \implies t \cdot \sigma : \iota \; in \; \mathcal{T}(F,V)$

The formalization of when a term matches another is straightforward:

definition ... **where** $l \; matches \; t = (\exists \; \sigma. \; t = l \cdot \sigma)$

and now we are ready to define the pattern completeness.

definition ... **where**
$pat\_complete\_lhss \; C \; D \; L = (\forall \; t \in \mathcal{B}(C,D,\emptyset). \; \exists l \in L. \; l \; matches \; t)$

5.2 Formalization of the Algorithm – Set Layer

The set based formalization is the one that is the furthest away from an executable version. Interestingly, it also deviates quite a bit from the textual description of the algorithm. Still, it is useful for proving that completeness of pattern problems is not altered by $\Rrightarrow$.

There are some deviations from the textual description that we like to mention.

First, we do not introduce the special problems $\bot_{mp}$ and $\top_{pp}$, e.g., by using an option-type. Instead, we split each set of inference rules in two parts, e.g., the matching problem transformation relation $\to$ into relation $\to_s$ that modifies an existing problem and into predicate *mp_fail* that leads to the special problem $\bot_{mp}$. In this way, the representation of matching and pattern problems stays simple, i.e., they are just (sets of) sets of pairs of terms.

Second, we change all $\uplus$-operators in the textual description into $\cup$-operators. This simplifies the reasoning for the refinements in the upcoming layers, but introduces nontermination. For instance, if $P = \{\top_{pp}\}$ then $P = \{\top_{pp}\} \cup P \Rrightarrow P$. Giving up on termination at this layer, we also join (instantiate) and (instantiate'): the formalization contains only one rule for instantiation at this layer, and this rule has no side-condition; i.e., it is always possible to instantiate $\{pp\}$ by $\mathsf{Inst}(pp, x)$ for any x.

Third, the formalization contains a notion of well-formedness for matching and pattern problems. In detail, the algorithm is formulated within a context that fixes a set S of sorts. Well-formedness enforces that the variables that occur in the problems only use sorts in S. Many of the properties are only proven for well-formed problems, and it is additionally proven that well-formedness is preserved by the transformations. Well-formedness does not enforce that the sets in a problem are finite; this is another source of nontermination on this layer.

We provide some example Isabelle snippets that formalize the relations $\to$ and $\Rightarrow$, illustrating the first two kinds of deviations. Here *insert a A* is Isabelle's notation for $\{a\} \cup A$.

inductive ... **where** $mp \to_s mp$
| $length \; ts = length \; ls \implies insert \; (Fun \; f \; ts, \; Fun \; f \; ls) \; mp \to_s set \; (zip \; ts \; ls) \cup mp$
| $x \notin \bigcup \; (vars \; `\; snd \; `\; mp) \implies insert \; (t, \; Var \; x) \; mp \to_s mp$

inductive *mp_fail* :: $('f,'v,'s) match\_problem\_set \to bool$
  where $(f,length \; ts) \neq (g,length \; ls) \implies mp\_fail \; (insert \; (Fun \; f \; ts, \; Fun \; g \; ls) \; mp)$
| `(* further inference rule for clash' *)`

inductive ... **where** *mp* $\rightarrow_s$ *mp'* $\Longrightarrow$ *insert mp pp* $\Rightarrow_s$ *insert mp' pp*
| *mp_fail mp* $\Longrightarrow$ *insert mp pp* $\Rightarrow_s$ *pp*

The relation $\Rrightarrow$ is formalized similarly. The main result of this layer is that $\Rrightarrow$ preserves pattern completeness on well-formed pattern problem sets, in Isabelle, *wf_pats*.

theorem *P_step_set_pcorrect*:
P $\Rrightarrow_s$ *P'* $\Longrightarrow$ *wf_pats P* $\Longrightarrow$ *pats_complete P* $\longleftrightarrow$ *pats_complete P'*

The most challenging rule was (failure') as detailed in the previous section. On the other hand, the most tedious one was (instantiate), which looks rather obvious on paper, but required 140 lines in our formalization.

5.3 Formalization of the Algorithm – Multiset Layer

On the next layer we use finite multisets to represent the algorithm. This layer is the one that is closest to the textual description and we fully prove Theorem 13 for this representation. The design of the formalization is as follows.

Concerning the relationship between textual and formalized version of the algorithm, we keep the deviation of splitting the inference rules from the previous layer, so that there is no need for the special problems $\bot_{mp}$ and $\top_{pp}$. Since a multiset union operation corresponds to a $\uplus$-operation on sets, there is no deviation at this point anymore. However, we require one further inference rule for matching problems whose necessity does not arise when working with sets. Since a multiset can have multiple occurrences of the same element, we need an explicit inference rule that is capable of deleting duplicates. To this end, we add the rule

$$\{(t, \ell), (t, \ell)\} \cup mp \rightarrow \{(t, \ell)\} \cup mp \qquad \text{(duplicate)}$$

on the multiset layer, which is then simulated by a new identity rule $mp \rightarrow mp$ on the set layer.

Partial correctness of $\Rrightarrow$ on this layer is obtained via the partial correctness result of $\Rrightarrow$ from the previous layer by proving a refinement property: The multiset-based implementation can be simulated by the set-based one.

The major new property that is added on this level is a formal proof of termination by closely following the textual proof.

We arrive at a formal version of Theorem 13 that looks quite similar to the textual one. Here, $\Rrightarrow$ in Isabelle refers to the multiset representation of $\Rrightarrow$, *SN* is strong normalization, i.e., termination, *pats_mset* converts from the multiset representation of pattern problems into the set representation, $\{\#\}$ is the empty multiset, and *bottom_mset* is representing $\bot_P$ as the multiset $\{\emptyset\}$.

theorem *SN_P_step*: *SN* $\Rrightarrow$
theorem *P_step*:
assumes *wf_pats* (*pats_mset P*) **and** (*P*,*Q*) $\in \Rrightarrow^!$
shows *Q* = $\{\#\}$ $\wedge$ *pats_complete* (*pats_mset P*)
$\vee$ *Q* = *bottom_mset* $\wedge$ $\neg$ *pats_complete* (*pats_mset P*)

5.4 Formalization of the Algorithm – List Layer

In the final layer we provide an executable version of the algorithm, by refining the multiset based version. To this end, we switch from multisets to lists; we turn the inductive inference rules into a recursive function definition; and we specify the order in which the inference rules will be applied. Our list-based implementation is split into several phases.

In the first phase, we exhaustively apply rules (decompose), (duplicate), (clash) and (clash'). Moreover, we organize the representation of the matching problems as follows.

- We store a list of pairs that have the structure $(x, f(\dots))$, i.e., those pairs on which (instantiate) is applicable.
- We store another list of pairs $([t_1, \dots, t_n], x)$ such that the matching problem contains all pairs $(t_1, x), \dots, (t_n, x)$, and no combination t_i and t_j results in a clash, and the list $[t_1, \dots, t_n]$ is distinct.
- We further store a Boolean flag whether the matching problem satisfies the condition of (instantiate') or not.

In the second phase, we apply rule (match) exhaustively and try to apply (failure'). Both of these steps can efficiently be implemented based on the previously described representation of matching problems.

Finally, if nothing else is applicable, then in the third phase we invoke (instantiate) or (instantiate'), with a preference on the former. In order to create fresh variables for the application of these rules, we assume that these variables are just numbers, and use a global index n which is incremented whenever a fresh variable is required.

These three phases are then iterated in a recursive function until a normal form is reached. By induction on $\Rrightarrow$, it is shown that the list-based implementation refines the multiset version of $\Rrightarrow$. Hence, partial correctness is easily transferred from the previous layer.

There is some additional glue-code required to get the final algorithm.

- We need to compute a high-enough value for the initial variable index n.
- We need to check the prerequisite that was stated at the beginning of this paper, namely that indeed all sorts are inhabited: $\{t \mid t : \iota \in \mathcal{T}(\mathcal{C})\} \neq \emptyset$. To this end, we verify a standard marking algorithm that computes the set of inhabited sorts: initially no sort is marked as inhabited, and whenever $c : \iota_1 \times \cdots \times \iota_n \to \iota_0 \in \mathcal{C}$ is a constructor and all sorts $\iota_1, \dots, \iota_n$ are marked, then also ι_0 is marked as inhabited. Finally, exactly the inhabited sorts are marked.
- We further require a function that determines whether a sort is finite or infinite; also here we verify a marking algorithm: initially no sort is marked as being finite, and whenever there is a sort ι where all constructors of that sort only have input sorts that are marked, then also ι is marked as finite. Finally, exactly the finite sorts are marked.[3] Interestingly, the dual approach (marking of infinite sorts whenever a recursive constructor is detected) is not so straight-forward, because sorts might be mutual recursive without direct recursion.

We finally provide a few wrapper functions that invoke the main decision procedure and get rid of its preconditions. For instance, for pattern completeness of programs (represented by their lhss) we obtain an algorithm *decide_pat_complete_lhss*.

theorem *decide_pat_complete_lhss*:
assumes *decide_pat_complete_lhss C D lhss = return b*
shows *b = pat_complete_lhss (map_of C) (map_of D) (set lhss)*

The algorithm will report an error on invalid input, e.g., if not all sorts are inhabited, or if the list of constructors C or the list of defined symbols D contain conflicting sort informations. If no such error is reported then the return value will be a Boolean b, and b is the completeness property of the set of lhss.

[3] The Isabelle formalization actually slightly deviates from this representation. It starts from the set of potentially infinite sorts and then iteratively removes the finite sorts.

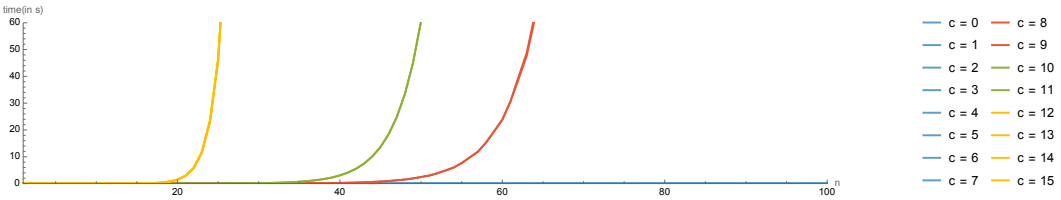


Figure 1 Timing of our algorithm for each configuration $c \in \{0, \ldots, 15\}$ on different sizes $n \in \{1, \ldots, 100\}$ using a timeout of 60 seconds.

6 Experiments

In order to evaluate the efficiency of our decision procedure, we use the following set of TRSs. They are similar to test programs that are used to show exponential behavior of match compilers for functional programming languages [16, Example 6].

▶ **Example 14.** We define TRS $\mathcal{R}_{c,n}$ for different configurations $c \in \{0, \ldots, 15\}$ and different sizes n. All TRSs use only one sort, namely the Booleans with constructors T and F, and there is only one defined symbol g. The TRS $\mathcal{R}_{c,n}$ consists of $2n + 1$ many rules and g has arity $2n$. We do not provide a full formal definition of $\mathcal{R}_{c,n}$, but instead illustrate the lhss of $\mathcal{R}_{c,n}$ for $c \in \{0, 1, 2, 4\}$ and $n = 3$, where each occurrence of _ represents a fresh variable.

$c = 0$	$c = 1$	$c = 2$	$c = 4$
$\mathsf{g}(\mathsf{F}, \_, \mathsf{F}, \_, \mathsf{F}, \_)$	$\mathsf{g}(\mathsf{F}, \mathsf{F}, \mathsf{F}, \_, \_, \_)$	$\mathsf{g}(\mathsf{F}, \_, \mathsf{F}, \_, \mathsf{F}, \_)$	$\mathsf{g}(\mathsf{T}, \mathsf{T}, \_, \_, \_, \_)$
$\mathsf{g}(\mathsf{T}, \mathsf{T}, \_, \_, \_, \_)$	$\mathsf{g}(\mathsf{T}, \_, \_, \mathsf{T}, \_, \_)$	$\mathsf{g}(\mathsf{T}, \mathsf{T}, \_, \_, \_, \_)$	$\mathsf{g}(\mathsf{T}, \mathsf{F}, \_, \_, \_, \_)$
$\mathsf{g}(\mathsf{T}, \mathsf{F}, \_, \_, \_, \_)$	$\mathsf{g}(\mathsf{T}, \_, \_, \mathsf{F}, \_, \_)$	$\mathsf{g}(\_, \_, \mathsf{T}, \mathsf{T}, \_, \_)$	$\mathsf{g}(\_, \_, \mathsf{T}, \mathsf{T}, \_, \_)$
$\mathsf{g}(\_, \_, \mathsf{T}, \mathsf{T}, \_, \_)$	$\mathsf{g}(\_, \mathsf{T}, \_, \_, \mathsf{T}, \_)$	$\mathsf{g}(\_, \_, \_, \_, \mathsf{T}, \mathsf{T})$	$\mathsf{g}(\_, \_, \mathsf{T}, \mathsf{F}, \_, \_)$
$\mathsf{g}(\_, \_, \mathsf{T}, \mathsf{F}, \_, \_)$	$\mathsf{g}(\_, \mathsf{T}, \_, \_, \mathsf{F}, \_)$	$\mathsf{g}(\mathsf{T}, \mathsf{F}, \_, \_, \_, \_)$	$\mathsf{g}(\_, \_, \_, \_, \mathsf{T}, \mathsf{T})$
$\mathsf{g}(\_, \_, \_, \_, \mathsf{T}, \mathsf{T})$	$\mathsf{g}(\_, \_, \mathsf{T}, \_, \_, \mathsf{T})$	$\mathsf{g}(\_, \_, \mathsf{T}, \mathsf{F}, \_, \_)$	$\mathsf{g}(\_, \_, \_, \_, \mathsf{T}, \mathsf{F})$
$\mathsf{g}(\_, \_, \_, \_, \mathsf{T}, \mathsf{F})$	$\mathsf{g}(\_, \_, \mathsf{T}, \_, \_, \mathsf{F})$	$\mathsf{g}(\_, \_, \_, \_, \mathsf{T}, \mathsf{F})$	$\mathsf{g}(\mathsf{F}, \_, \mathsf{F}, \_, \mathsf{F}, \_)$

The 16 configurations are obtained by combining 4 different kinds to arrange the arguments of g with 4 different orders of the rules. For the argument orders of g we choose the following four alternatives, visualized by reordering the arguments of the first lhs of $\mathcal{R}_{0,n}$: $\mathsf{g}(\mathsf{F}, \_, \mathsf{F}, \_, \ldots, \mathsf{F}, \_)$ or $\mathsf{g}(\_, \mathsf{F}, \_, \mathsf{F}, \ldots, \_, \mathsf{F})$ or $\mathsf{g}(\mathsf{F}, \mathsf{F}, \ldots, \mathsf{F}, \_, \_, \ldots, \_)$ or $\mathsf{g}(\_, \_, \ldots, \_, \mathsf{F}, \mathsf{F}, \ldots, \mathsf{F})$. Concerning the order of the rules, we either put the first rule of $\mathcal{R}_{0,n}$ to the front position ($c = 0$) or to the last position ($c = 4$); and we either group the other $2n$ rules in n blocks of size 2 ($c = 0$) or in 2 blocks of size n ($c = 2$).

Since for a given n all configurations result in the same set of lhss (modulo symmetries), the question of pattern completeness should be equally hard for all configurations. However, since our implementation of the decision procedure has a fixed order in which rules are applied and in which variables are instantiated, there is quite a different behavior in the execution time, cf. Figure 1.

Choosing $c \in \{0, \ldots, 7\}$ results in a low execution time, where the corresponding blue line in Figure 1 is not distinguishable from the x-axis: for instance, deciding pattern completeness of $\mathcal{R}_{c,100}$ is finished within 0.06 seconds. However, for $c \in \{8, \ldots, 15\}$ an exponential behavior becomes visible, where $c = \{8, 9\}$, $c = \{10, 11\}$, and $c = \{12, 13, 14, 15\}$ each have similar behavior.

We further compare our decision procedure with three other algorithms.

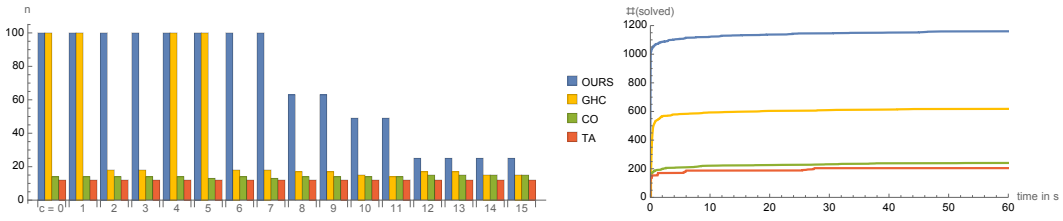


Figure 2 Left: Increase n for each $c \in \{0, \ldots, 15\}$ until $n = 100$ or 60 seconds timeout is reached. Right: Number of solved $\mathcal{R}_{c,n}$ instances for $(c, n) \in \{0, \ldots, 15\} \times \{1, \ldots, 100\}$ within time limit.

- GHC: we encode $\mathcal{R}_{c,n}$ as a Haskell program and use the `ghc` Haskell compiler and ask it to warn about incomplete patterns. To be more precise we invoke `ghc` with parameters `-c -Wincomplete-patterns -fmax-pmcheck-models=1000000` where the latter number is chosen in such a way that no approximation is occurring.[4]
- CO: we run the complement algorithm on the TRSs, taking the implementation that is available in the ground confluence prover AGCP [1].
- TA: given $\mathcal{R}_{c,n}$ we define two tree automata $\mathcal{A}_n$ (with transitions $\mathsf{F} \to \mathsf{bool}$, $\mathsf{T} \to \mathsf{bool}$ and $\mathsf{g}(\mathsf{bool}, \ldots, \mathsf{bool}) \to \mathsf{accept}$) and $\mathcal{B}_{c,n}$ (using four common transitions $\mathsf{F} \to \mathsf{bool}$, $\mathsf{F} \to \mathsf{false}$, $\mathsf{T} \to \mathsf{bool}$, $\mathsf{T} \to \mathsf{true}$, and one further transition for each rule, e.g., $\mathsf{g}(\mathsf{bool}, \mathsf{bool}, \mathsf{bool}, \mathsf{bool}, \mathsf{true}, \mathsf{false}) \to \mathsf{accept}$ for the last rule of $\mathcal{R}_{0,3}$) so that pattern completeness of $\mathcal{R}_{c,n}$ is equivalent to the language inclusion problem $\mathcal{L}(\mathcal{A}_n) \subseteq \mathcal{L}(\mathcal{B}_{c,n})$. We then invoke the tree automaton library of FORT-h [13] to decide this inclusion property.

We ran experiments where for each configuration c and each algorithm we increased n until either $\mathcal{R}_{c,n}$ for $n = 100$ was successfully analyzed, or until there was a 60 seconds timeout when handling $\mathcal{R}_{c,n}$. In Figure 2 we display the maximal values of n (left) and the cumulative solved instances plot in the style of SAT-competition [6] (right).

The diagram clearly shows that our new decision procedure outperforms all other three algorithms on the test suite. Interestingly, also in GHC there is a strong dependence on the configuration, i.e., the execution time varies between polynomial and exponential. This is different for TA and CO: these algorithms always resulted in exponential behavior, independent of the choice of c.

For further details on the experiments we refer to the website with supplementary material.

7 Conclusion and Future Work

We developed a new decision procedure to decide pattern completeness that is not restricted to the left-linear case. The corresponding verified implementation is faster than previous approaches, in particular it performs better than the complement algorithm and tree automata based methods.

We see some opportunities for future work. First, one can integrate an improved strategy to select variables for instantiation, in particular since permutations in the input cause severe differences in runtime. One can also try to further improve the implementation, e.g., by

[4] When invoking `ghc`, it does not only check pattern completeness, but also compiles the program. However, the compilation time is negligible in our experiments. On all programs where GHC was successful, the compilation time decreased to below 0.5 seconds when turning off the pattern completeness check.

following suggestions of Sestoft [16, Section 7.5] such as the integration of memoization. The latter corresponds to a rule $\{pp, pp\} \cup P \Rrightarrow \{pp\} \cup P$ to detect and eliminate duplicate pattern problems. Second, one might add counter-example generation into the formalization and into the verified implementation. Third, it remains open whether a similar syntax directed decision procedure for quasi-reducibility can be designed, i.e., where matching may occur in arbitrary subterms. Finally, one might consider an extension where it is allowed to add structural axioms to some symbols such as associativity and commutativity.

References

1 Takahito Aoto and Yoshihito Toyama. Ground confluence prover based on rewriting induction. 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 33:1–33:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICS.FSCD.2016.33.

2 Adel Bouhoula. Simultaneous checking of completeness and ground confluence for algebraic specifications. *ACM Trans. Comput. Log.*, 10(3):20:1–20:33, 2009. doi:10.1145/1507244.1507250.

3 Adel Bouhoula and Florent Jacquemard. Sufficient completeness verification for conditional and constrained TRS. *J. Appl. Log.*, 10(1):127–143, 2012. doi:10.1016/j.jal.2011.09.001.

4 Hubert Comon. Sufficient completness, term rewriting systems and "anti-unification". In Jörg H. Siekmann, editor, *8th International Conference on Automated Deduction, Oxford, England, July 27 - August 1, 1986, Proceedings*, volume 230 of *Lecture Notes in Computer Science*, pages 128–140. Springer, 1986. doi:10.1007/3-540-16780-3_85.

5 Stephan Falke and Deepak Kapur. Rewriting induction + linear arithmetic = decision procedure. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Automated Reasoning - 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings*, volume 7364 of *Lecture Notes in Computer Science*, pages 241–255. Springer, 2012. doi:10.1007/978-3-642-31365-3_20.

6 Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda. SAT competition 2020. *Artif. Intell.*, 301:103572, 2021. doi:10.1016/J.ARTINT.2021.103572.

7 Deepak Kapur, Paliath Narendran, Daniel J. Rosenkrantz, and Hantao Zhang. Sufficient-completeness, ground-reducibility and their complexity. *Acta Informatica*, 28(4):311–350, 1991. doi:10.1007/BF01893885.

8 Deepak Kapur, Paliath Narendran, and Hantao Zhang. On sufficient-completeness and related properties of term rewriting systems. *Acta Informatica*, 24(4):395–415, 1987. doi:10.1007/BF00292110.

9 Cynthia Kop. Quasi-reductivity of logically constrained term rewriting systems. *CoRR*, 2017. arXiv:1702.02397.

10 Alexander Krauss. Partial and nested recursive function definitions in higher-order logic. *J. Autom. Reasoning*, 44(4):303–336, 2010. doi:10.1007/s10817-009-9157-2.

11 Alain Laville. Lazy pattern matching in the ML language. In Kesav V. Nori, editor, *Foundations of Software Technology and Theoretical Computer Science, Seventh Conference, Pune, India, December 17-19, 1987, Proceedings*, volume 287 of *Lecture Notes in Computer Science*, pages 400–419. Springer, 1987. doi:10.1007/3-540-18625-5_64.

12 Azeddine Lazrek, Pierre Lescanne, and Jean-Jacques Thiel. Tools for proving inductive equalities, relative completeness, and omega-completeness. *Inf. Comput.*, 84(1):47–70, 1990. doi:10.1016/0890-5401(90)90033-E.

13 Aart Middeldorp, Alexander Lochmann, and Fabian Mitterwallner. First-order theory of rewriting for linear variable-separated rewrite systems: Automation, formalization, certification. *J. Autom. Reason.*, 67(2):14, 2023. doi:10.1007/S10817-023-09661-7.

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

15 Uday S. Reddy. Term rewriting induction. In Mark E. Stickel, editor, *10th International Conference on Automated Deduction, Kaiserslautern, FRG, July 24-27, 1990, Proceedings*, volume 449 of *Lecture Notes in Computer Science*, pages 162–177. Springer, 1990. doi:10.1007/3-540-52885-7_86.

16 Peter Sestoft. MK pattern match compilation and partial evaluation. In Olivier Danvy, Robert Glück, and Peter Thiemann, editors, *Partial Evaluation, International Seminar, Dagstuhl Castle, Germany, February 12-16, 1996, Selected Papers*, volume 1110 of *Lecture Notes in Computer Science*, pages 446–464. Springer, 1996. doi:10.1007/3-540-61580-6_22.

17 Christian Sternagel and René Thiemann. First-order terms. *Archive of Formal Proofs*, February 2018. , Formal proof development. URL: https://isa-afp.org/entries/First_Order_Terms.html.

18 Jean-Jacques Thiel. Stop losing sleep over incomplete data type specifications. In Ken Kennedy, Mary S. Van Deusen, and Larry Landweber, editors, *Conference Record of the Eleventh Annual ACM Symposium on Principles of Programming Languages, Salt Lake City, Utah, USA, January 1984*, pages 76–82. ACM Press, 1984. doi:10.1145/800017.800518.

Commutation Groups and State-Independent Contextuality

Samson Abramsky
Department of Computer Science, University College London, London, UK

Şerban-Ion Cercelescu
Department of Computer Science, University of Oxford, Oxford, UK

Carmen-Maria Constantin
Department of Computer Science, University College London, London, UK

Abstract

We introduce an algebraic structure for studying state-independent contextuality arguments, a key form of quantum non-classicality exemplified by the well-known Peres-Mermin magic square, and used as a source of quantum advantage. We introduce *commutation groups* presented by generators and relations, and analyse them in terms of a string rewriting system. There is also a linear algebraic construction, a directed version of the Heisenberg group. We introduce *contextual words* as a general form of contextuality witness. We characterise when contextual words can arise in commutation groups, and explicitly construct non-contextual value assignments in other cases. We give unitary representations of commutation groups as subgroups of generalized Pauli n-groups.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Contextuality, state-independence, quantum mechanics, Pauli group, group presentations, unitary representations

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.28

Funding *Samson Abramsky*: EPSRC EP/V040944/1 Resources in Computation, UKRI 10050493 FoQaCiA
Carmen-Maria Constantin: UKRI 10050493 FoQaCiA

1 Introduction

Contextuality is a key form of non-classicality in quantum mechanics, and is the source of quantum advantage in a range of settings, including measurement-based quantum computation [17] and shallow circuits [8, 7]. In classical physics, observable quantities have well-defined values independently of which measurements are performed. This is contradicted by the predictions of quantum mechanics [13], as verified in numerous experiments [6, 10]. These say that values can only be assigned *locally*, in measurement contexts, *i.e.* with respect to sets of measurements which can be performed together, providing observational "windows" of classical information on the quantum system. These windows may overlap, and will agree on their overlaps (*local consistency*), but it is not possible, on pain of logical contradiction, to glue all these pieces of information together (*global inconsistency*).

The strongest form of this phenomenon is *state-independent contextuality*, where the structure of the observables dictates that contextuality arises for any state. The most famous example of this phenomenon is the Peres-Mermin magic square [14], which is constructed from the 2-qubit Pauli group[1]:

[1] We recall the definition of the Pauli group in the Appendix.

$$\begin{array}{ccccc} XI & - & IX & - & XX \\ | & & | & & | \\ IZ & - & ZI & - & ZZ \\ | & & | & & | \\ XZ & - & ZX & - & YY \end{array}$$

Here XI denotes the 2-qubit operator $\sigma_x \otimes I$, and similarly for the other entries. One can now calculate that the operators in each row and column pairwise commute, and hence form a valid measurement context. Moreover, the product of each of the rows, and of the first two columns, is II; while the product of the third column is $-II$.

We shall now see how to recognize contextuality in this example. The key point is that this can be done *a priori*, independently of any observational data. We ask if there is an assignment of outcomes $v : \mathfrak{X} \to \mathbb{Z}_2$, where $\mathfrak{X}$ is the set of operators in the table, subject to the conditions that

1. if p and q commute, then $v(pq) = v(p) + v(q)$.
2. $v(II) = 0$ and $v(-II) = 1$.

Such an assignment is called a *non-contextual value assignment*. If no such assignment exists, this yields an example of *contextuality*. We call this *state-independent*, since it arises purely at the level of the operators in the table, independently of any state.

Note that we only require the homomorphism condition (1) for *commuting* operators, which correspond to observables that can be performed together, in a common context. This is the key idea introduced by Kochen and Specker in their seminal work on contextuality [13].

Now assume for contradiction that such an assignment exists. We obtain the following set of equations over $\mathbb{Z}_2$ from the above table, one for each row and each column:

$$\begin{array}{rclcrcl} a+b+c & = & 0 & \qquad & a+d+g & = & 0 \\ d+e+f & = & 0 & & b+e+h & = & 0 \\ g+h+i & = & 0 & & c+f+i & = & 1 \end{array} \tag{1}$$

Here a is a variable corresponding to $v(XI)$, etc.

Since each variable appears twice in the left hand sides, summing over them yields 0, while summing over the right hand sides yields 1. This yields the required contradiction.

The justification for assuming the partial homomorphism condition comes from the quantum case, where if A and B are commuting observables and ψ is a common eigenvector of A and B, with eigenvalue v for A and w for B, then ψ is an eigenvector for AB with eigenvalue vw. Also, II has the unique eigenvalue $+1$, and $-II$ the unique eigenvalue -1.[2]

We now wish to abstract from the specifics of the Pauli group, and understand the general structure which makes such arguments possible. This leads us to introduce the notion of *commutation group*, to which we now turn.

2 Commutation groups

The idea behind commutation groups is that they are built freely from prescribed commutation relations on a set of generators. Commutation relations play a fundamental role in quantum mechanics, the canonical example being the commutation relation between position and momentum (see e.g. [11]): $[p, q] = i\hbar\mathbb{1}$. We can think of a commutation relation as saying that two elements commute *up to a prescribed scalar*. For this to make sense in a group theoretic

[2] Note that $\{+1, -1\}$ under multiplication is an isomorphic representation of $\mathbb{Z}_2$, with 0 corresponding to $+1$ and 1 to -1 under the mapping $i \mapsto (-1)^i$.

context, we need an action of a suitable (classical, hence abelian) group of scalars or "phases" on the group we are constructing. We are interested here in finite group constructions, so we shall work over the finite cyclic groups $\mathbb{Z}_d$, $d \geq 2$.

Given a finite set $\mathfrak{X}$ of generators, we define a *commutator matrix* to be a map $\mu : \mathfrak{X}^2 \to \mathbb{Z}_d$ which is skew-symmetric, meaning that $\mu(x, y) = -\mu(y, x)$ for all $x, y \in X$. We also assume that $\mu(x, x) = 0$ for all $x \in \mathfrak{X}$.[3]

We shall describe the construction of commutation groups from commutator matrices in two ways: by generators and relations, and by a linear algebraic construction. Both are useful, and convey different intuitions.

2.1 Commutation groups by generators and relations

We briefly review the standard notion of *presentation of a monoid by generators and relations* $\langle \mathfrak{X} \mid R \rangle$. We form the free monoid $\mathfrak{X}^*$, and quotient it by the congruence induced from the relations $R \subseteq \mathfrak{X}^* \times \mathfrak{X}^*$. Explicitly, we define a symmetric relation $\underset{R}{\longleftrightarrow} \subseteq \mathfrak{X}^* \times \mathfrak{X}^*$ by $s \underset{R}{\longleftrightarrow} t$ iff there is $(u, v) \in R \cup R^{-1}$ such that, for some $w_1, w_2 \in \mathfrak{X}^*$, $s = w_1 u w_2$, and $t = w_1 v w_2$. We then take the reflexive transitive closure $\overset{*}{\underset{R}{\longleftrightarrow}}$. This is a monoid congruence, and the quotient $M = \mathfrak{X}^* / \overset{*}{\underset{R}{\longleftrightarrow}}$ is the presented monoid.

Notation. We write relations as $u \doteq v$. We write the empty sequence, which forms the identity element of the free monoid, as 1.

Given a commutator matrix $\mu : \mathfrak{X}^2 \to \mathbb{Z}_d$, we define a set of relations R_{G} over the generators $\mathfrak{X} \sqcup \mathbb{Z}_d$ (using $\sqcup$ for disjoint union), where we write J_k for the generator corresponding to $k \in \mathbb{Z}_d$, and:

- We have relations $R_\mu := \{xy \doteq J_{\mu(x,y)} yx \mid x, y \in \mathfrak{X}\}$.
- We have $R_J := \{J_0 \doteq 1\} \cup \{J_k J_{k'} \doteq J_{k+k'} \mid k, k' \in \mathbb{Z}_d\} \cup \{J_k x \doteq x J_k \mid x \in \mathfrak{X}, k \in \mathbb{Z}_d\}$.
- We have $R_d := \{x^d \doteq 1 \mid x \in \mathfrak{X}\}$.
- Finally, $R_{\mathsf{G}} := R_\mu \cup R_J \cup R_d$.

The resulting monoid $\mathsf{G}(\mu) := \langle \mathfrak{X} \sqcup \mathbb{Z}_d \mid R_{\mathsf{G}} \rangle$ is in fact a group, since every generator has an inverse. We call it the *commutation group* generated by μ.

The J-relations ensure that there is an isomorphic copy of $\mathbb{Z}_d$ in the centre of the group. The key relations are the commutation relations $xy \doteq J_{\mu(x,y)} yx$. Note that these are *directional*, since by skew-symmetry of μ, if $\mu(x, y) = k$, then $yx \doteq J_{-k} xy$. Thus moving x right past y has the opposite "cost" to moving x left past y. This suggests that we can analyze $\mathsf{G}(\mu)$ by a directed *string rewriting system*.

To do this, we fix a linear ordering $x_1 < \cdots < x_n$ on $\mathfrak{X}$.[4] Relative to this ordering, elements of $\mathsf{G}(\mu)$ can be represented as ordered multisets over $\mathfrak{X}$ with multiplicities strictly less than d, together with a "global phase" from $\mathbb{Z}_d$. Explicitly, we define $\mathcal{N}$ to be the set of all expressions $J_k x_1^{k_1} \cdots x_n^{k_n}$, with $k \in \mathbb{Z}_d$, and $0 \leq k_i < d$, $1 \leq i \leq n$. There is an evident bijection $\mathcal{N} \cong \mathbb{Z}_d \times \mathbb{Z}_d^n$. Thus $\mathcal{N}$ has cardinality d^{n+1}.

We now define a string rewriting system on $\mathfrak{X} \sqcup \mathbb{Z}_d$, obtained by orienting a subset of the relations R_{G}, determined by the chosen linear order on $\mathfrak{X}$:

- $\to_\mu := \{xy \to J_{\mu(x,y)} yx \mid x > y\}$.
- $\to_J := \{J_0 \to 1\} \cup \{J_k J_{k'} \to J_{k+k'} \mid k, k' \in \mathbb{Z}_d\} \cup \{x J_k \to J_k x \mid x \in \mathfrak{X}, k \in \mathbb{Z}_d\}$.
- $\to_d := \{x^d \to 1\}$.
- $\to_{\mathsf{G}} := \to_\mu \cup \to_d \cup \to_J$.

[3] Note that if d is even, this does not follow automatically from skew-symmetry.
[4] As we shall see, the choice of ordering is immaterial, leading to isomorphic results.

This induces a relation on $(\mathfrak{X} \sqcup \mathbb{Z}_d)^*$ by $s \to t$ iff for some $u \to_{\mathsf{G}} v$, for some $w_1, w_2 \in (\mathfrak{X} \sqcup \mathbb{Z}_d)^*$, $s = w_1 u w_2$, and $t = w_1 v w_2$.

▶ **Theorem 1.** *The rewrite system $\to_{\mathsf{G}}$ is confluent and normalizing. The set of normal forms is $\mathcal{N}$ (up to identification of J_0 and 1).*

Proof. Given a word $s \in (\mathfrak{X} \sqcup \mathbb{Z}_d)^*$, we define:

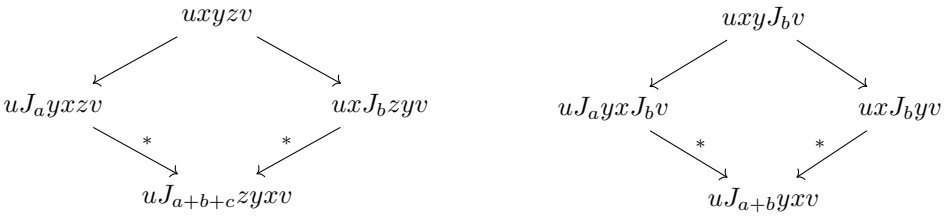
- An $\mathfrak{X}$-inversion in s is (u, v, w, x, y) such that $s = uxvyw$, and $x > y$.
- A J-inversion in s is (u, v, w, x, k) such that $s = uxvJ_k w$.

We define a function $\varphi : (\mathfrak{X} \sqcup A)^* \to \mathbb{N} \times \mathbb{N} \times \mathbb{N}$ by $\varphi(s) = (n, m, l)$, where n is the number of $\mathfrak{X}$-inversions in s, m is the number of J-inversions, and l is the length of s.

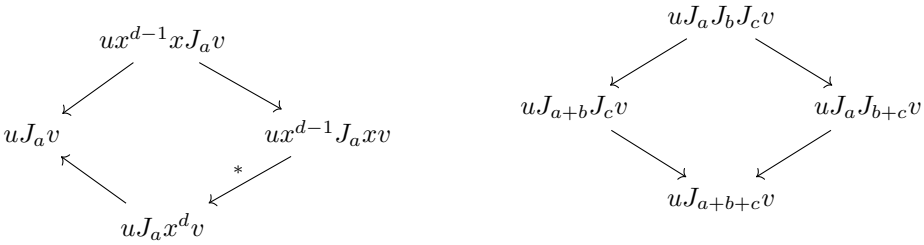
We now observe that for each rewrite $s \to t$ in the above system $\to_{\mathsf{G}}$, $\varphi(s) \succ \varphi(t)$ in the lexicographic ordering on $\mathbb{N} \times \mathbb{N} \times \mathbb{N}$. Indeed, the μ relations decrease the number of $\mathfrak{X}$-inversions, the J-commutation rule decreases the number of J-inversions while not increasing the number of $\mathfrak{X}$-inversions, and the remaining rules decrease length while not increasing the number of inversions. Since this ordering is well-founded, it follows that $\to_{\mathsf{G}}$ is normalizing.

By Newman's Lemma, it now suffices to show that $\to_{\mathsf{G}}$ is weakly confluent. This is verified straightforwardly by examining the critical pairs.

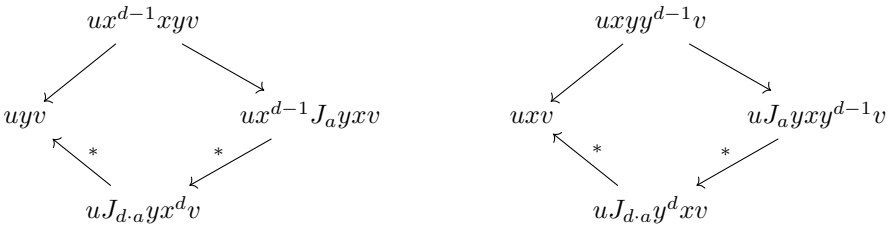
Firstly, consider $x > y > z$, $\mu(x, y) = a$, $\mu(y, z) = b$, $\mu(x, z) = c$:



Next, two cases involving J-generators:



Finally:



Note that $d \cdot a = 0 \mod d$, justifying the final legs. ◀

By virtue of this theorem, we can define a function $\theta : (\mathfrak{X} \sqcup \mathbb{Z}_d)^* \to \mathcal{N}$, which returns the normal form of a word. Note that, if $w \to^* w'$, then by confluence, $\theta(w) = \theta(w')$.

We can use this function to define an equivalence on $(\mathfrak{X} \sqcup \mathbb{Z}_d)^*$ by $s \simeq t$ iff $\theta(s) = \theta(t)$. This equivalence is in fact a congruence, since if $\theta(u) = s = \theta(u')$ and $\theta(v) = t = \theta(v')$, then, using confluence, $\theta(uv) = \theta(st) = \theta(u'v')$.

▶ **Proposition 2.** *For all $s, t \in (\mathfrak{X} \sqcup \mathbb{Z}_d)^*$, $s \simeq t$ iff $s \doteq t$.*

Proof. The left-to-right implication follows immediately since $\to_{\mathsf{G}} \subset R_{\mathsf{G}}$. For the converse, it suffices to show that $s \simeq t$ for all relations $s \doteq t$ in R_{G}, since $\doteq$ is the least congruence containing these relations.

Consider firstly $xy \doteq J_k yx$, where $k = \mu(x, y)$. There are two cases:

1. If $x < y$, then $xy \in \mathcal{N}$, and $J_k yx \to J_k J_{-k} xy \to^* xy$.
2. If $x > y$, then $J_k yx \in \mathcal{N}$, and $xy \to J_k yx$.

The other relations are verified similarly. ◀

We now define a monoid with carrier $\mathcal{N}$. Note that 1 and J_k, $k \in \mathbb{Z}_d$, are in $\mathcal{N}$. We define the multiplication by $u \cdot v := \theta(uv)$.

▶ **Proposition 3.** *$(\mathcal{N}, \cdot, 1)$ is a monoid.*

Proof. We need to verify associativity. This follows from

$$\theta(\theta(uv)w) = \theta(uvw) = \theta(u\theta(vw)) \tag{2}$$

which in turn follows from confluence. ◀

We now define a map $h : \mathsf{G}(\mu) \to \mathcal{N}$ by $h([w]) = \theta(w)$.

▶ **Theorem 4.** *The map h is well-defined, and is a monoid isomorphism $h : \mathsf{G}(\mu) \cong \mathcal{N}$.*

Proof. If $u \doteq v$, then by Proposition 2, $\theta(u) = \theta(v)$. Thus h is well-defined. The fact that it preserves multiplication follows from $\theta(uv) = \theta(\theta(u)\theta(v))$, which follows from confluence. If $w \in \mathcal{N}$, then $h([w]) = \theta(w) = w$. Thus h is surjective. Finally, if $h([u]) = h([v])$, then $u \simeq v$, so by Proposition 2, $[u] = [v]$. ◀

We now come to a key property for applications to contextuality.

▶ **Theorem 5.** *The internal $\mathbb{Z}_d$-action given by the J-generators is faithful: if $J_k \doteq J_{k'}$ in $\mathsf{G}(\mu)$, then $k = k'$.*

Proof. This is immediate from the isomorphic representation given by $\mathcal{N}$, since if $k \neq k'$, J_k and $J_{k'}$ are distinct normal forms. ◀

The parameter d plays a double role in the commutation groups, defining the order of the generators by the relations $x^d = 1$, and also the abelian "phase group" $\mathbb{Z}_d$ acting on the commutation group. We used this double role of d in proving confluence for the rewriting system. This assumption is in fact necessary to obtain a confluent system with a faithful action, as the following example shows.

▶ **Example 6.** We assume the relations $x^d \doteq 1$ for the generators. Consider the word $w \equiv yxy^{d-1}x^{d-1}$, and let $a = \mu(x, y)$. Then $w \doteq J_{(d-1)\cdot a} y^d x^d \doteq J_{(d-1)\cdot a}$, and also $w \doteq J_{-a} xy^d x^{d-1} \doteq J_{-a} x^d \doteq J_{-a}$. Thus to maintain confluence and faithfulness of the action, we require $(d-1) \cdot a = -a$, and hence $d \cdot a = 0$.

Comparison with solution groups

In [9] another way of abstracting from the Peres-Mermin square and similar constructions is pursued, leading to the introduction of *solution groups*. These groups are specified by sets of equations of similar form to (1). The generators appearing together in an equation, and only these, are specified to commute. These groups are shown in [9] to control the question of whether there is a quantum realization for these equations. Importantly, this is shown to be equivalent to the existence of quantum perfect strategies for Alice-Bob non-local games.

A remarkable result of Slofstra [19] shows that solution groups, even over $\mathbb{Z}_2$, are extremely expressive (*i.e.* "wild"). Every finitely presentable group can be embedded in a solution group. It follows immediately that the word problem for solution groups, and hence the quantum realization questions, are undecidable.

By contrast, commutation groups are highly tractable. By Theorem 4, they are always finite. From the proof of termination in Theorem 1, we see that reduction to normal form, and hence the decision procedure for the word problem, is at most quadratic in the length of the word. As we shall see later, every commutation group admits a faithful unitary representation.

2.2 Linear algebraic construction of commutation groups

The characterization of commutation groups in Theorem 4 suggests another description. We shall now use the fact that $\mathbb{Z}_d$ is not just an abelian group, but a commutative ring with unit. We can write a commutator matrix, with a chosen order on the set of generators, as an $n \times n$ matrix with entries in $\mathbb{Z}_d$. We write $\mathfrak{so}(n, \mathbb{Z}_d)$ for the set of all $n \times n$ commutator matrices (skew-symmetric and zero on the diagonal) over $\mathbb{Z}_d$. Given a commutator matrix μ, we write $\check{\mu}$ for its lower triangular part, so that $\mu = \check{\mu} - \check{\mu}^{\mathsf{T}}$.

An $n \times n$ matrix M over $\mathbb{Z}_d$ defines a bilinear form on the free $\mathbb{Z}_d$-module $\mathbb{Z}_d^n$, by $M(\vec{k}, \vec{l}) := \vec{k}^{\top} M \vec{l}$. Now given $\mu \in \mathfrak{so}(\mathbb{Z}_d, n)$, we define a group $H(\mu)$ with carrier $\mathbb{Z}_d \times \mathbb{Z}_d^n$. The group product is defined by

$$(k, \vec{k}) \cdot (l, \vec{l}) \;=\; (k + l + \check{\mu}(\vec{k}, \vec{l}),\, \vec{k} + \vec{l}).$$

Thus it is precisely the phase factor $\check{\mu}(\vec{k}, \vec{l})$ which makes the group non-commutative.

The associativity of the product follows from bilinearity. The unit is $(0, 0)$. The inverse of $(k, \vec{k})$ is $(-k - \check{\mu}(\vec{k}, -\vec{k}), -\vec{k})$.

▶ **Proposition 7.** *For any $\mu \in \mathfrak{so}(n, \mathbb{Z}_d)$, $H(\mu) \cong \mathsf{G}(\mu)$.*

Proof. By Theorem 4, the carriers are in evident bijection: $J_k x_1^{k_1} \cdots x_n^{k_n} \leftrightarrow (k, (k_1, \ldots, k_n))$. We just have to check that the group product is preserved. The only non-imediate part of this is to check that the phase factors agree.

Suppose in $\mathcal{N}$ we have normal forms with vector parts $u = x_1^{k_1} \cdots x_n^{k_n}$ and $v = x_1^{l_1} \cdots x_n^{l_n}$. To combine them into $\theta(uv)$, with vector part $x_1^{k_1+l_1} \cdots x_n^{k_n+l_n}$, we must move l_1 copies of x_1 over k_n copies of x_n, each with a cost of $\mu(x_n, x_1)$; and similarly for the occurrences of $x_{n-1}, \ldots, x_2$ in u, with total cost $\sum_{i>1} k_i \mu(x_i, x_1) l_1$. A similar analysis applies to the occurrences of $x_2, \ldots, x_{n-1}$ in v, leading to a total cost of $\sum_{i>j} k_i \mu(x_i, x_j) l_j$. This is exactly $\vec{k}^{\top} \check{\mu} \vec{l} = \check{\mu}(\vec{k}, \vec{l})$. ◀

Note that we would get the same result if we moved the vector part of u rightwards over the vector part of v. The choice of left/right orientation is just a convention. On the other hand, the use of $\check{\mu}$ rather than μ is significant. As we will see in the next section, using μ would render the structure useless for our purpose of analyzing contextuality.

However, we do retrieve μ as the group-theoretic commutator in $H(\mu)$.

▶ **Proposition 8.** *Given $g = (k, \vec{k}), h = (l, \vec{l}) \in H(\mu)$, their group theoretic commutator is given by $[g, h] := ghg^{-1}h^{-1} = (\check{\mu}(\vec{k}, \vec{l}) - \check{\mu}(\vec{l}, \vec{k}), 0) = (\mu(\vec{k}, \vec{l}), 0)$. In terms of $\mathsf{G}(\mu)$, for $u, v \in X^*$, $[u, v] \doteq J_{k-l}$, where $\theta(uv) \doteq J_k w$, $\theta(vu) \doteq J_l w$.*

As Proposition 7 makes clear, commutation groups are very close to the (discrete version of) the Heisenberg or Heisenberg-Weyl groups [18], and their close relatives the Pauli groups. The novelty lies mainly in our combinatorial mode of presentation of commutation groups, which we will make use of in our analysis of contextuality arguments. It should be noted, though, that the direct equivalent of the usual Heisenberg group construction in our setting would be to use the full commutator matrix μ.[5] As we have mentioned, using μ would yield a non-isomorphic construction, which would not be useful for analyzing contextuality. This perhaps suggests that we can think of commutation groups as a *directed* version of Heisenberg groups.

3 Contextuality arguments in commutation groups

The commutation group has an evident short exact sequence

$$\mathbf{0} \longrightarrow \mathbb{Z}_d \xrightarrow{\ i\ } H(\mu) \xrightarrow{\ \pi_2\ } \mathbb{Z}_d^n \longrightarrow \mathbf{0} \tag{3}$$

where $i(k) = (k, 0)$. This says that it is a non-abelian group extension of $\mathbb{Z}_d^n$ by $\mathbb{Z}_d$. The image of $\mathbb{Z}_d$ lies in the centre of $H(\mu)$, so the extension is central. Because of the non-commutativity of $H(\mu)$, it is easy to see that there is no left-splitting of this extension, *i.e.* a homomorphism $l : H(\mu) \to \mathbb{Z}_d$ such that $l \circ i = \mathsf{id}_{\mathbb{Z}_d}$. One could say that this simple observation is essentially a form of von Neumann's much criticised No-Go theorem for hidden variables [15]. The point of the criticism is that it is not reasonable to ask for a splitting which preserves non-commuting products.

Following Kochen-Specker [13] and the huge literature on ensuing developments, we want to consider only assignments to *observationally accessible contexts*, *i.e.* those constructed from commuting products. A general setting for capturing this idea is provided by *compatible monoids*, introduced in [1] with different terminology. A compatible monoid is a structure $(M, \odot, \cdot, 1)$, where $\odot$ is a reflexive, symmetric relation on M, of "compatibility" or "comeasurability", and $\cdot : \odot \to M$ is a partial binary operation with domain $\odot \subseteq M^2$, such that:

- $x \odot y \ \Rightarrow\ x \cdot y = y \cdot x$,
- $x \odot 1$ for all $x \in M$, and $x \cdot 1 = x$,
- if $x \odot y$, $x \odot z$, and $y \odot z$, then $x \odot (y \cdot z)$ and $(x \cdot y) \odot z$, and $(x \cdot y) \cdot z = x \cdot (y \cdot z)$.

Homomorphisms of compatible monoids are maps which preserve the compatibility relation, and the monoid operations when defined.

Any monoid M defines a compatible monoid with the same carrier, with $x \odot y$ iff $xy = yx$ in M. We will be interested in the compatible submonoid of M generated by a set $S \subseteq M$. This is the least set T containing $S \cup \{1\}$, and such that, whenever $u, v \in T$ and $u \odot v$, then $u \cdot v \in T$. In particular, we will apply this to $\mathsf{G}(\mu)$ with respect to the generators $\mathfrak{X} \sqcup \mathbb{Z}_d$. We will write $\mathsf{C}(\mu)$ for this compatible submonoid of $\mathsf{G}(\mu)$.

[5] There is a notion of polarized Heisenberg group, but this is isomorphic to the usual presentation.

In terms of $H(\mu)$, we can identify the generators as follows: $k \in \mathbb{Z}_d$ can be identified with the scalar $(k, 0)$, while the generators $\mathfrak{X}$ can be identified with the standard basis E of the free module $\mathbb{Z}_d^n$, $x_i \leftrightarrow e_i := [\underbrace{0, \ldots, 0}_{i-1}, 1, \underbrace{0, \ldots, 0}_{n-i}]^\top$.

We obtain a short exact sequence for $\mathsf{C}(\mu)$:

$$\mathbf{0} \longrightarrow \mathbb{Z}_d \xrightarrow{\ i\ } \mathsf{C}(\mu) \xrightarrow{\ p\ } P \longrightarrow \mathbf{0} \tag{4}$$

Here p is the restriction of the second projection to $\mathsf{C}(\mu)$, and P is its image.

A non-contextual value assignment for the commutation group $\mathsf{G}(\mu)$ is exactly a left splitting of this short exact sequence: *i.e.* a homomorphism $l : \mathsf{C}(\mu) \to \mathbb{Z}_d$ such that $l \circ i = \mathsf{id}_{\mathbb{Z}_d}$. If no such left splitting exists, then we say that $\mathsf{G}(\mu)$ exhibits *state-independent contextuality*.

3.1 Contextual words

We can distill the essential features of "parity proofs" such as the one given for the Peres-Mermin square into a notion of *contextual word*, which provides a witness for state-independent contextuality. This notion was introduced, somewhat informally, in the concrete context of Pauli groups over qubits in [12], but can be formulated generally for any commutation group $\mathsf{G}(\mu)$. A contextual word for $\mathsf{G}(\mu)$ is given by a triple (w, β, k) such that:

- $w \in \mathfrak{X}^+$.
- The number of occurrences of each generator $x \in \mathfrak{X}$ in w is a multiple of d.
- β is a bracketing of w, witnessing that it is in $\mathsf{C}(\mu)$.
- $w \doteq J_k$, where $k \neq 0$.

Bracketings are defined inductively by

$$\beta \in \mathsf{BE} \ ::= \ x \mid (\beta_1, \beta_2).$$

We define $\partial : \mathsf{BE} \to \mathfrak{X}^+$ by $\partial(x) = x$, $\partial(\beta_1, \beta_2) = \partial(\beta_1)\partial(\beta_2)$. If $\partial(\beta) = w$, then w is the word bracketed by β. A bracketing β provides a witness for $w = \partial(\beta) \in \mathsf{C}(\mu)$ if, for every (β_1', β_2') occurring in β, with $u = \partial(\beta_1')$ and $v = \partial(\beta_2')$, $uv \doteq vu$ in $\mathsf{G}(\mu)$.

▶ **Proposition 9.** *If there is a contextual word for $\mathsf{G}(\mu)$, then it is state-independently contextual.*

Proof. If (w, β, k) is a contextual word over S, assume for a contradiction that $l : \mathsf{C}(\mu) \to \mathbb{Z}_d$ is a non-contextual value assignment, *i.e.* a left splitting of (4). The bracketing β witnesses that $w \in \mathsf{C}(\mu)$. By the homomorphism property, $l(w) = \sum_i l(x_i)$, where $w = x_1 \cdots x_n$. Since each $x \in \mathfrak{X}$ occurs with multiplicity kd in w for some $k \geq 0$, $l(w) = 0 \pmod d$. However, we also have $w \doteq J_k$, so we must have $l(w) = l(J_k) = k \neq 0$, yielding the required contradiction. ◀

▶ **Example 10.** Consider the following commutator matrices over $\mathbb{Z}_2$

$$\mu_1 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \qquad \mu_2 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix} \qquad \mu_3 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

with generators $a < b < c < d$. Then $(w_i, \beta_i, 1)$ is a contextual word for $\mathsf{G}(\mu_i)$, with

$$\begin{array}{lcllcl} w_1 & = & abdccabd & \beta_1 & = & ((ab)(dc))((ca)(bd)) \\ w_2 & = & bdccaabd & \beta_2 & = & (b(dc))((ca)((ab)d)) \\ w_3 & = & dcabbadc & \beta_3 & = & (d(ca))(b(((ba)d)c)) \end{array}$$

▶ **Example 11.** We show that the Peres-Mermin square arises in commutation groups.

Firstly, we show that the tensor product construction underlying the extension of the Pauli group to $\mathbf{Pauli}_n$ has a simple form in terms of group presentations. Given a commutator matrix $\mu : \mathfrak{X}^2 \to \mathbb{Z}_d$, we define $\mu_2 : (\mathfrak{X} + \mathfrak{X})^2 \to \mathbb{Z}_d$ by $\mu_2(x_i, y_i) = \mu(x, y)$, $\mu_2(x_i, y_j) = 0$, $i \neq j$. Thus elements in different copies of $\mathfrak{X}$ commute with each other.

We now consider the commutator matrix with $\mu(x, y) = 1$ for generators x, y. We can define the following Peres-Mermin square over $\mathsf{G}(\mu_2)$:

$$\begin{array}{ccccc} x_1 & — & x_2 & — & x_1x_2 \\ | & & | & & | \\ y_2 & — & y_1 & — & y_1y_2 \\ | & & | & & | \\ x_1y_2 & — & y_1x_2 & — & J_1(x_1y_1)(x_2y_2) \end{array}$$

We can verify that exactly the same algebraic properties hold for this square as in the concrete example: each row and column pairwise commutes, the product of each row and the first two columns is 1, the product of the third column is -1 (or more pedantically, J_1 in additive notation).

We can extract a contextual word from this construction: $((x_1y_2)(y_1x_2))((x_1x_2)(y_1y_2))$. Up to dropping the J_1 factor, and interchanging the commuting pair x_2y_1, this can be read off from product of the bottom row of the square.

This provides a more succinct contextuality witness than the usual parity proof, which amounts to taking the product of all the rows and columns.

A similar treatment can be given of the Mermin star [14].

3.2 Comparison with other Heisenberg groups

We can now see why taking the more standard Heisenberg group construction, defined exactly as for $H(\mu)$, but using μ rather than $\check{\mu}$, would not be suitable for our purposes. Let us denote the construction using μ rather than $\check{\mu}$ by $H^+(\mu)$. By Proposition 8, the commutator in $H(\mu)$ is μ, which means that we can have commuting products $gh = hg$ with non-zero but equal phase factors, which is clearly essential for contextual words to exist. By contrast, the commutator in $H^+(\mu)$ is easily seen to be 2μ, which means in Z_2 that *all products commute*, while in odd orders, no products commute.

4 No state-independent contextuality in odd characteristics

We shall now show that contextual words can only exist over $\mathbb{Z}_d$ if d is even. Moreover, we shall explicitly describe the non-contextual value assignments which exist when d is odd.

In order to prove these results, we will analyze the structure of inversions in bracketed words.

Notation. In this section, we will deal exclusively with non-empty words over the generators, $w \in \mathfrak{X}^+$. We will also write formal sums in variables $v_{x,y}$ to stand in for values of the commutator matrix $\mu(x, y)$. We will use the following notation. If $S = \{\lambda_i\}_{i \in I}$ is a family of inversions, then $\sum S := \sum_{i \in I} v_{x_i, y_i}$, where λ_i is an inversion between x_i and y_i, $x_i > y_i$.

Given a word s, we write $\mathcal{I}(s)$ for the set of inversions in s. Given words s, t, $\mathcal{I}(s, t)$ is the set of inversions between s and t, *i.e.* the set of all (w_1, x, w_2, y) such that w_1x is a prefix of s, w_2y is a prefix of t, and $x > y$.

The following is immediate.

▶ **Lemma 12.** *For all words s, t,*

$$\sum \mathcal{I}(st) \;=\; \sum \mathcal{I}(s) + \sum \mathcal{I}(t) + \sum \mathcal{I}(s,t).$$

In this notation, the equation forcing the global phase factor for a word s to be k is

$$\sum \mathcal{I}(s) \;=\; k.$$

Also, given words s, t, we define the "formal commutator" of s and t to be

$$[\![s,t]\!] \;:=\; \sum \mathcal{I}(s,t) - \sum \mathcal{I}(t,s).$$

The equation forcing s and t to commute is $\sum \mathcal{I}(st) - \sum \mathcal{I}(ts) = 0$. By the previous lemma, this is equivalently written as $[\![s,t]\!] = 0$.

▶ **Lemma 13.** *Let $s^\dagger$ be the reverse of a word s. Then $\sum \mathcal{I}(s,t) \;=\; \sum \mathcal{I}(s^\dagger, t^\dagger)$.*

Proof. Note that $s^\dagger$ defines the same multiset of occurrences of generators as s, so there will be a bijection between the inversions in $\mathcal{I}(s,t)$ and those in $\mathcal{I}(s^\dagger,t^\dagger)$, inducing the same multiset of variables $v_{x,y}$. ◀

We now consider bracketings of words. Given a bracketing β, we define the multiset $\Phi(\beta)$ by

$$\Phi(x) = \varnothing, \qquad \Phi(\beta_1,\beta_2) = \{\!\{(\partial(\beta_1),\partial(\beta_2))\}\!\} \uplus \Phi(\beta_1) \uplus \Phi(\beta_2).$$

Given a word s with bracketing β, we can write $\Phi(\beta)$ as a family $\{(s_i,t_i)\}_{i\in I}$ of adjacent subwords of s corresponding to subexpressions of the full bracketing.

▶ **Lemma 14.** *With notation as above, let $s^\dagger$ be the reverse of s. Then*

$$\sum_{i\in I} [\![s_i,t_i]\!] \;=\; \sum \mathcal{I}(s) \;-\; \sum \mathcal{I}(s^\dagger). \tag{5}$$

Proof. By induction on the length of s. If $s = x$, then the sums on both sides of (5) are empty, and we have the equation $0 = 0$.

In the inductive case, suppose the top-level bracketing of s is $s = uv$. We can write the bracketings of u and v as families $\{(u_j,u'_j)\}_{j\in J}$, $\{(v_k,v'_k)\}_{k\in K}$. Then, applying the induction hypothesis:

$$\begin{aligned}\textstyle\sum_{i\in I}[\![s_i,t_i]\!] &= [\![u,v]\!] \;+\; \textstyle\sum_j [\![u_j,u'_j]\!] \;+\; \sum_k [\![v_k,v'_k]\!] \\ &= \left(\textstyle\sum \mathcal{I}(u,v) - \sum \mathcal{I}(v,u)\right) \;+\; \left(\sum \mathcal{I}(u) - \sum \mathcal{I}(u^\dagger)\right) \;+\; \left(\sum \mathcal{I}(v) - \sum \mathcal{I}(v^\dagger)\right).\end{aligned}$$

By Lemma 12, $\sum \mathcal{I}(s) \;=\; \sum \mathcal{I}(uv) \;=\; \sum \mathcal{I}(u,v) + \sum \mathcal{I}(u) + \sum \mathcal{I}(v)$. Since $s^\dagger = v^\dagger u^\dagger$, applying Lemma 12 again yields $\sum \mathcal{I}(s^\dagger) \;=\; \sum \mathcal{I}(v^\dagger,u^\dagger) + \sum \mathcal{I}(v^\dagger) + \sum \mathcal{I}(u^\dagger)$. Applying Lemma (13) and rearranging terms yields (5). ◀

▶ **Lemma 15.** *Let w be a word in which each generator x occurs n_x times, modulo d. Then*

$$\sum \mathcal{I}(w) + \sum \mathcal{I}(w^\dagger) = \sum_{x<y} n_y n_x v_{yx}$$

In particular, when each generator occurs a multiple of d times, we have $\sum \mathcal{I}(w) \;=\; -\sum \mathcal{I}(w^\dagger)$.

Proof. In order to count the number of occurrences of the variable v_{yx}, consider each occurrence of y within w. For the i^{th} occurrence we can write $w = u_i y v_i$ and for each x such that $x < y$, the m_i occurrences of x in v_i will yield m_i inversions in w, while the occurrences of x in u_i will yield n_i inversions in $w^\dagger$. Thus, the total multiplicity of $v_{x,y}$ in $\sum \mathcal{I}(w)$ will be $\sum_i m_i$, where i ranges over occurrences of y in w. Similarly, the total multiplicity of $v_{x,y}$ in $\sum \mathcal{I}(w^\dagger)$ will be $\sum_i n_i$. Since for each i, $m_i + n_i = n_x$ we will have the overall multiplicity

$$n_{yx} = \sum_{i=1}^{n_y} m_i + \sum_{i=1}^{n_y} n_i = \sum_{i=1}^{n_y} n_x = n_y n_x.$$

When each n_x is a multiple of d we therefore have $\sum \mathcal{I}(w) + \sum \mathcal{I}(w^\dagger) = 0 \pmod d$. ◀

▶ **Theorem 16.** *If (w, β, k) is a contextual word over $\mathbb{Z}_d$, then d is even.*

Proof. Since (w, β, k) is contextual, we have $[\![s, t]\!] = 0$ for all bracketed subexpressions (s, t) in β. Hence summing over all such subexpressions yields $\sum_i [\![s_i, t_i]\!] = 0$. By Lemma 14, this implies that $\sum \mathcal{I}(w) - \sum \mathcal{I}(w^\dagger) = 0$. Applying Lemma 15 yields $2\sum \mathcal{I}(w) = 0$. The contextuality of (w, β, k) forces $\sum \mathcal{I}(w) = k$, where $k \neq 0$. We can only have a non-zero solution of $2k = 0 \pmod d$ if d is even. ◀

▶ **Theorem 17.** *If w_1 and w_2 are words in $\mathfrak{X}^+$ formed out of commuting products and which have the same multiset of generators, modulo d, then if d is odd their overall commutation factors are equal.*

Proof. Since w_1 and w_2 are formed out of commuting products, each of the formal commutators corresponding to the sub-expressions of w_1 and w_2 is equal to zero, hence $\sum \mathcal{I}(w_i) = \sum \mathcal{I}(w_i^\dagger)$. From Lemma 15 it follows that

$$2\sum \mathcal{I}(w_i) = \sum_{x<y} n_y^i n_x^i \cdot v_{yx}$$

The right hand side of this equation is the same for w_1 and w_2, since the number n_x^i of occurrences of each generator is equal modulo d in the two words. Since d is odd, the equation $2x = k \pmod d$ has a unique solution for any $k \in \mathbb{Z}_d$, and $\sum \mathcal{I}(w_1) = \sum \mathcal{I}(w_2)$. ◀

▶ **Theorem 18.** *Let μ be a commutator matrix over $\mathbb{Z}_d$. If d is odd, there is a non-contextual value assignment $\upsilon : \mathsf{C}(\mu) \to \mathbb{Z}_d$.*

Proof. We use the vector representation of $\mathsf{C}(\mu) \subseteq \mathsf{G}(\mu)$. Define $S := \{\vec{k} \in \mathbb{Z}_d^n \mid \exists k.\, (k, \vec{k}) \in \mathsf{C}(\mu)\}$. By Lemma 17, there is a unique $\varphi(\vec{k}) \in \mathbb{Z}_d$ such that every word $w \in \mathfrak{X}^+$ which can be formed by commuting products and evaluates to a normal form $\theta(w)$ with corresponding vector part $\vec{k} \in S$ has global phase factor $\varphi(\vec{k})$. Thus if w is such a word, $\varphi(\vec{k}) = \sum \mathcal{I}(w)$. By Theorem 16, $\varphi(\mathbf{0}) = 0$. Given $(k, \vec{k}) \in \mathsf{C}(\mu)$, we define $\upsilon : (k, \vec{k}) \mapsto k - \varphi(\vec{k})$. Clearly this is left-inverse to the inclusion $\iota : \mathbb{Z}_d \to \mathsf{C}(\mu)$. We must verify the homomorphism condition. Given a commuting product $(k, \vec{k}) \cdot (l, \vec{l}) = (k + l + \breve{\mu}(\vec{k}, \vec{l}), \vec{k} + \vec{l})$ in $\mathsf{C}(\mu)$, we must show that

$$(k - \varphi(\vec{k})) + (l - \varphi(\vec{l})) \;=\; (k + l + \breve{\mu}(\vec{k}, \vec{l})) - \varphi(\vec{k} + \vec{l}),$$

i.e. that $\varphi(\vec{k} + \vec{l}) = [\varphi(\vec{k}) + \varphi(\vec{l}) + \breve{\mu}(\vec{k}, \vec{l})]$. Taking words s, t evaluating to $\vec{k}$, $\vec{l}$, this is $\sum \mathcal{I}(st) = \sum \mathcal{I}(s) + \sum \mathcal{I}(t) + \sum \mathcal{I}(s, t)$, *i.e.* Lemma 12. ◀

5 Contextuality in even characteristics

We now show that contextual words exist in abundance in even characteristics. Firstly, we characterize the circumstances under which non-contextual value assignments *do* arise.

Given $S \subseteq \mathsf{C}(\mu)$, we define $\mathcal{Z}(S) := \{a \in S \mid \forall b \in S.\, a \odot b\}$. A *graph* will always mean a reflexive undirected graph, *i.e.* a set of vertices with a reflexive, symmetric relation.

A *cluster graph* is a coproduct (disjoint union) of complete graphs. Equivalently, it is a graph in which the adjacency relation is transitive, so that the maximal cliques are the equivalence classes, and hence disjoint, with no adjacencies between them.

We will show that, if $(\mathsf{C}(\mu) \setminus \mathcal{Z}(\mathsf{C}(\mu)), \odot)$ is a cluster graph, every empirical model over $\mathsf{C}(\mu)$ has global sections, which are exactly non-contextual value assignments.

We briefly review what we need of empirical models; for further details, see [4, 3]. A maximal clique over the graph $(\mathsf{C}(\mu), \odot)$ is a (total) commutative sub-monoid of $\mathsf{C}(\mu)$: closure under products is implied by maximality. Moreover, it contains $\mathcal{Z}(\mathsf{C}(\mu))$. Let $\mathscr{M}$ be the set of maximal cliques. Note that the union of this family is $\mathsf{C}(\mu)$.

A (possibilistic) empirical model over $\mathsf{C}(\mu)$ assigns to each $C \in \mathscr{M}$ a non-empty set of homomorphisms $s : C \to \mathbb{Z}_d$ which split the inclusion $\mathbb{Z}_d \hookrightarrow C$. We write $\{e_C\}_{C \in \mathscr{M}}$ for this family of sets of homomorphisms. The family is moreover required to satisfy the following *local consistency* property: for all $C, C' \in \mathscr{M}$, $e_C|_{C \cap C'} = e_{C'}|_{C \cap C'}$, where e.g

$$e_C|_{C \cap C'} := \{s|_{C \cap C'} \mid s \in e_C\}.$$

We say that such an empirical model is non-contextual (in the sense of not strongly contextual [4]) if there exists a *global section*: a homomorphism $s : \mathsf{C}(\mu) \to \mathbb{Z}_d$ such that $s|_C \in e_C$ for all $C \in \mathscr{M}$. Such a global section is necessarily a left splitting, and hence a non-contextual value assignment for $\mathsf{C}(\mu)$.

▶ **Theorem 19.** *If $(\mathsf{C}(\mu) \setminus \mathcal{Z}(\mathsf{C}(\mu)), \odot)$ is a cluster graph, then every empirical model over $\mathsf{C}(\mu)$ is non-contextual.*

Proof. Let $N := \mathsf{C}(\mu) \setminus \mathcal{Z}(\mathsf{C}(\mu)), Z := \mathcal{Z}(\mathsf{C}(\mu))$. Each maximal clique of $(\mathsf{C}(\mu), \odot)$ is of the form $C \sqcup Z$, where C is a maximal clique of $(N, \odot)$. Let e be an empirical model, and consider $s \in e_{C \sqcup Z}$ for $C \sqcup Z \in \mathscr{M}$. We can write $s = [s_C, s_Z] : C \sqcup Z \to \mathbb{Z}_d$. By the local consistency property for e, for any $C' \neq C$ maximal in $(N, \odot)$, there is $s_{C'} : C' \to \mathbb{Z}_d$ such that $s' = [s_{C'}, s_Z] : C' \sqcup Z \to \mathbb{Z}_d \in e_{C' \sqcup Z}$. Morever, as C', C'' range over maximal cliques of $(N, \odot)$, since $C' \cap C'' = \varnothing$, $s' = [s_{C'}, s_Z]$ is compatible with $s'' = [s_{C''}, s_Z]$, *i.e.* $s'|_Z = s_Z = s''|_Z$. Thus we obtain a pairwise compatible family of sections $\{[s_C, s_Z]\}_{C \sqcup Z \in \mathscr{M}}$.

Since $\mathscr{M}$ covers $\mathsf{C}(\mu)$, this family determines a unique function $s : \mathsf{C}(\mu) \to \mathbb{Z}_d$. We must check the homomorphism condition. This holds because whenever $g \odot h$, $\{g, h\} \subseteq C$ for some $C \in \mathscr{M}$, hence $s(gh) = s_C(gh) = s_C(g) + s_C(h) = s(g) + s(h)$. ◀

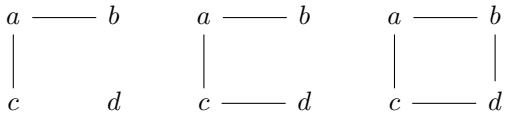
Note that in the last part of the argument, we were verifying the sheaf property for the cover $\mathscr{M}$ over the presheaf of left splittings on cliques in $(\mathsf{C}(\mu), \odot)$.

One remaining question is whether empirical models over $\mathsf{C}(\mu)$ actually exist.[6] We shall discuss unitary representations of commutation groups in the next section. Given a quantum realization of the associated measurements, we can always obtain an empirical model by applying any quantum state.

[6] The issue is whether we can have a non-empty model satisfying the local consistency conditions.

5.1 Positive results

We now assume that $(\mathsf{C}(\mu) \setminus \mathcal{Z}(\mathsf{C}(\mu)), \odot)$ is *not* a cluster graph, which means that there are elements a, b, c such that $a \odot b$, $a \odot c$, but not $b \odot c$. Since a is not in $\mathcal{Z}(\mathsf{C}(\mu))$, there must be some d such that not $a \odot d$. Allowing for the various possibilities for commutativity of d with b and c, up to relabelling this gives us the following three compatibility graphs [12]:



(6)

5.1.1 The $\mathbb{Z}_2$ case

In the case where μ is a commutator matrix over $\mathbb{Z}_2$, we can give a definitive characterisation of contextuality in $\mathsf{C}(\mu)$. This follows similar lines to [12], in the general setting of commutation groups.

▶ **Theorem 20.** *If μ is a commutator matrix over $\mathbb{Z}_2$, then the following are equivalent:*

1. *$\mathsf{C}(\mu)$ is contextual.*
2. *There are contextual words over $\mathsf{C}(\mu)$.*
3. *The graph $(\mathsf{C}(\mu), \odot)$ contains one of the graphs in (6) as an induced sub-graph.*

Proof. The implication (2) ⇒ (1) is Proposition 9. By contraposition, (1) ⇒ (3) follows from Theorem 19. Now assume (3). If a, b, c, d are generators, the matrices μ_i given in Example 10 correspond to the graphs in (6), and the corresponding contextual words given in the Example show that (2) holds. Otherwise, these elements arise as commuting products, each of which can be described by a suitably bracketed word. If any of these words has global phase factor 1, they are already contextual words. Otherwise, we can substitute them into the words given in Example 10 to obtain contextual words. ◀

5.1.2 Beyond $\mathbb{Z}_2$: padding, splitting and variable changes

We can transfer contextual words from $\mathbb{Z}_2$ to $\mathbb{Z}_{2k}$, using the embedding $\mathbb{Z}_2 \rightarrowtail \mathbb{Z}_{2k}$ which sends 1 to k, which can be applied to a commutator matrix over $\mathbb{Z}_2$ to produce one over $\mathbb{Z}_{2k}$. If we take any of the contextual words w from Example 10, we can then perform a simple padding construction. We append $a^{2k-2}b^{2k-2}c^{2k-2}d^{2k-2}$ to w, and this produces a contextual word over $\mathbb{Z}_{2k}$.

Can we construct contextual words over $\mathbb{Z}_{2k}$ using matrix values other than 0 and k? By Theorem 16, the global phase factor for a contextual word over $\mathbb{Z}_{2k}$ must be k, but we may use other values from $\mathbb{Z}_{2k}$ in constructing the word. We can use a splitting construction to achieve this. We illustrate the idea with a simple example over $\mathbb{Z}_4$. Given the contextual word $((ab)(dc))((ca)(bd))$ from Example 10, we split the generator a into a_1 and a_2. We can use the commutator matrix

$$\mu = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 3 & 3 & 1 \\ 3 & 1 & 0 & 2 & 0 \\ 3 & 1 & 2 & 0 & 0 \\ 3 & 3 & 0 & 0 & 0 \end{bmatrix}$$

and obtain the contextual word $[((a_1a_2)b)(cd)][((a_1a_2)c)(bd)][a_1^2a_2^2b^2c^2d^2]$, using also the padding construction described previously.

5.1.3 Classification of contextuality for matrices in Darboux normal form

Our overall aim is to give a complete classification of which commutation groups $\mathsf{G}(\mu)$ admit contextual words. We shall achieve this for matrices μ in *Darboux normal form*, *i.e.* whose only non-zero entries occur in block 2×2 matrices on the main diagonal of the form

$$\begin{matrix} 0 & \lambda_i \\ -\lambda_i & 0 \end{matrix}$$

Given a matrix μ over $\mathbb{Z}_{2n}$, which is in Darboux normal form, it is possible to decide whether it supports contextual words by considering the parity, relative to n, of the non-zero entries of μ. By relative parity, we mean whether the power with which 2 appears in the prime factor decomposition of n is lower than the power with which it appears in the prime factor decomposition of each of the non-zero entries. Thus, if $n = n' \times 2^y$ and $\lambda = l \times 2^x$, where n' and l' are both odd integers, we say that λ is even relative to n if $x > y$ and that λ is odd relative to n if $x \leq y$.

Firstly, a preliminary lemma. We use the notation $m \vdots n$ to mean that m is divisible by n.

▶ **Lemma 21.** *If a word s is formed out of commuting products, and s_a and s_b denote the multiplicities with which a and b appear within s, then $s_a s_b \vdots 2^{k+1}$.*

Proof. We prove this by induction on the length of s. If s is length 1 then either s_a or s_b, or both, are zero, and the statement holds trivially. Assume that the statement holds for words of length less than or equal to L and let s be a word of length $L+1$. Then $s = uv$ for some u and v of length at most L and let u_a, u_b, v_a, v_b denote the respective multiplicities of a and b within u and v. Since u and v commute, we must have

$$(u_a v_b - u_b v_a) \vdots 2^{k+1}$$

By the inductive hypothesis $u_a u_b \vdots 2^{k+1}$ and $v_a v_b \vdots 2^{k+1}$ and therefore $u_a u_b v_a v_b \vdots 2^{2k+2}$. If $u_a v_b \not\vdots 2^{k+1}$ then $u_b v_a$ must be divisible by 2^{k+1} and this would contradict the commutativity condition. Hence both $u_a v_b$ and $u_b v_a$ are divisible by 2^{k+1}. This allows us to complete the inductive proof, as $s_a = u_a + v_a$ and $s_b = u_b + v_b$ and so the product $s_a s_b$ expands as a sum of terms which are each divisible by 2^{k+1}:

$$s_a s_b = u_a u_b + u_a v_b + u_b v_a + v_a v_b. \qquad \blacktriangleleft$$

▶ **Theorem 22.** *If μ is in Darboux normal form, then there is a contextual word over $\mathsf{G}(\mu)$ if and only if there are two non-zero entries above the main diagonal, $\lambda_i = l_i \times 2^{x_i}$ and $\lambda_j = l_j \times 2^{x_j}$ which are both odd relative to n.*

Proof. If we can find two non-zero entries above the main diagonal, $\lambda_i = l_i \times 2^{x_i}$ and $\lambda_j = l_j \times 2^{x_j}$ which are both odd relative to n and we denote their corresponding variables by a, b, c, d, then we can form the contextual word

$$((\underbrace{a \dots a}_{k}\ \underbrace{c \dots c}_{k})(bd))((\underbrace{a \dots a}_{k} d)(b \underbrace{c \dots c}_{k}))(\underbrace{a \dots a}_{m})(\underbrace{b \dots b}_{2n-2})(\underbrace{c \dots c}_{m})(\underbrace{d \dots d}_{2n-2})$$

where $k = n' \times 2^{y-x_j}$, $m = n' \times 2^{y-x_i}(2^{x_i+1} - 2)$ and $k + m = 2n$. It is straightforward to check that all the brackets commute and that the overall commutation factor is equal to

$$n' \times 2^{y-x_i} \times l_i \times 2^{x_i} = n \times l_i$$

which is equal to n modulo $2n$, since l_i is odd.

On the other hand, if all the non-zero entries of μ are even relative to n then we will not be able to get any word w with a non-zero commutation factor since, as we have shown in Section 4, the commutation factor of w must satisfy the equation

$$2\sum \mathcal{I}(w) = 0$$

In $\mathbb{Z}_{2n}$ the only non-zero solution to this equation is $\sum \mathcal{I}(w) = n = n' \times 2^y$ and since all the commutation variables in the matrix μ have a factor of 2 greater than y, any linear combination of them will also have a factor of 2 greater than y, and so will not yield a commutation factor equal to n modulo $2n$.

Finally, we can show that if only one non-zero entry is odd relative to n, while all the others are relatively even, then contextual words cannot exist. For simplicity of notation, we will show this for 4 generators but the proof, which is essentially a parity argument, works equally well for any number of generators. Let $n = n' \times 2^y$, as before, and let a and b denote the two variables whose corresponding entry in the commutation matrix is $m \times 2^{y-k}$ for some odd m and $k \geq 0$. By assumption, all other entries are of the form $m' \times 2^{y+1+t}$ for some odd m' and $t \geq 0$. Then any bracketed subexpression of the form

$$w = (a^{l_a} b^{l_b} c^{l_c} d^{l_d})(a^{r_a} b^{r_b} c^{r_c} d^{r_d})$$

commutes only if

$$2^{y-k} \times m(l_a r_b - l_b r_a) + 2^{y+1+t} \times m'(l_c r_d - l_d r_c) = N \times 2^{y+1} \times n'$$

Since the right hand side is a multiple of 2^{y+1} and the terms on the left hand side coming from the relatively even entries are also multiples of 2^{y+1} and m is odd, it follows that

$$(l_a r_b - l_b r_a) \vdots 2^{k+1}$$

By Lemma 21, $l_b r_a$ is divisible by 2^{k+1} and so the contribution to the overall commutation factor, which is

$$2^{y-k} \times m l_b r_a + 2^{y+1+t} \times m' l_d r_c$$

will also be a multiple of 2^{y+1}. Recall that the only possible non-zero value for the overall commutation factor is n modulo $2n$ which implies

$$\sum \mathcal{I}(w) = (2N+1)n = (2N+1)n' \times 2^y$$

and therefore a sum of terms which are divisible by 2^{y+1} cannot yield a non-zero overall commutation factor, which completes the proof. ◀

5.1.4 Reduction to Darboux normal form

Every commutator matrix can be reduced to one in Darboux normal form. This is standard over a field, but less obvious over $\mathbb{Z}_d$ for arbitrary d, so we include a proof.

Note that since μ plays the role of a bilinear form, if we wish to perform a change of basis preserving this form, we must perform the corresponding row and column operations on the matrix μ. These operations are encoded by an invertible base change matrix U; the resulting matrix $U^T \mu U$ is said to be *cogredient* to μ.

▶ **Lemma 23.** *Every commutation matrix μ is cogredient to a skew-symmetric matrix μ_d whose only non-zero entries occur directly above and below the main diagonal. We call this the standard form of the commutation matrix.*

Proof. We start by noting that the "swapping" matrix $U_{i,j}$ which is obtained by swapping the i^{th} and j^{th} columns of the identity matrix is self-inverse over $\mathbb{Z}_{2n}$ and

$$U_{i,j}^T \mu U_{i,j}$$

is the commutation matrix obtained from μ by first swapping the i^{th} and j^{th} rows of μ and then the i^{th} and j^{th} columns.

Similarly, the "adding" matrix $V_{i,j}^{\alpha}$ which is obtained by adding to the i^{th} column of the identity matrix α times the j^{th} column is invertible, its inverse being $V_{i,j}^{-\alpha}$. Hence the matrix

$$V_{i,j}^{\alpha T} \mu V_{i,j}^{\alpha}$$

is cogredient with μ and has the corresponding effect of adding α times the j^{th} row/column of μ to its i^{th} row/column.

Using these two types of cogredient operations it is possible, using Euclid's algorithm to change μ into a cogredient matrix μ_d with the desired property.

The first step is to consider the n^{th} row of the matrix μ:

$$\begin{array}{ccccc} \ddots & \vdots & \vdots & \vdots & \vdots \\ \dots & 0 & * & * & -c \\ \dots & * & 0 & * & -b \\ \dots & * & * & 0 & -a \\ \dots & c & b & a & 0 \end{array}$$

If it has k non-zero entries, we can use suitable swapping $U_{i,j}$ matrices to bring those entries to the right of all zero entries, ordered ascendingly. Then if $a = \mu(n, n-1) > \mu(n, n-2) = b$ we can use $V_{n-1,(n-2)}^{\alpha}$ and $U_{n-1,n-2}$ matrices to perform Euclid's algorithm on the bottom entries of these penultimate two columns, resulting in a cogredient matrix μ_1 with $\mu_1(n, n-2) = 0$ and $\mu_1(n, n-1) = gcd(a, b)$. For example, if the first step of the algorithm gives the decomposition $a = bq_1 + r_1$ then the matrix

$$\mu' = U_{n-1,n-2}^T V_{n-1,n-2}^{-q_1 T} \mu V_{n-1,n-2}^{-q_1} U_{n-1,n-2}$$

will have $\mu'(n, n-2) = r_1$ and $\mu'(n, n-1) = b$. If r_1 is non-zero, we can continue iterating the next steps of the algorithm, until eventually we reach μ_1.

The next step is to consider $c = \mu_1(n, n-3)$ and use suitable $U_{n-3,n-1}$ and $V_{n-1,n-3}^{\alpha}$ matrices to again perform Euclid's algorithm, resulting in a matrix μ_2 for which $\mu_2(n, n-2) = \mu_2(n, n-3) = 0$ and $\mu_2(n, n-1) = gcd(a, b, c)$. And we proceed to eliminate all the k next non-zero entries of the last row, thus leaving

$$\mu_{k-1}(n, n-1) = gcd(\mu(n, n-1), \mu(n, n-2), \dots, \mu(n, n-k))$$

as the only non-zero entry on the n^{th} row. And since cogredient operations result in skew-symmetric matrices, the only non-zero entry on the n^{th} column will also be the one above the main diagonal.

We can repeat these steps to clear out the t non-zero entries on row $n-1$, which are to the left of the $(n-1, n-2)$ position. This results in some matrix μ_{t-1} whose only nonzero entries on row $n-1$ are $\mu_{t-1}(n-1, n) = \mu_{k-1}(n-1, n)$ and

$$\mu_{t-1}(n-1, n-2) = gcd(\mu_{k-1}(n-1, n-2), \mu_{k-1}(n-1, n-3), \dots, \mu_{k-1}(n-1, n-t)).$$

We proceed similarly with the remaining rows, eventually resulting in a matrix μ_d in standard form. ◀

▶ **Theorem 24.** *Every commutation matrix μ is cogredient to a matrix μ_D in Darboux normal form, whose only non-zero entries occur in block 2×2 matrices on the main diagonal of the type*

$$\begin{matrix} 0 & \lambda_i \\ -\lambda_i & 0 \end{matrix}$$

Proof. From Lemma 23, we know that μ is cogredient to a matrix μ_d in standard form. We describe an algorithm which, given a 4×4 diagonal block of μ_d of the type

$$\begin{matrix} 0 & a & 0 & 0 \\ -a & 0 & b & 0 \\ 0 & -b & 0 & c \\ 0 & 0 & -c & 0 \end{matrix}$$

performs cogredient operations on μ_d to produce a block in Darboux normal form of the type

$$\begin{matrix} 0 & \lambda_1 & 0 & 0 \\ -\lambda_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \lambda_2 \\ 0 & 0 & -\lambda_2 & 0 \end{matrix}$$

Assume without loss of generality that the 4×4 block is in the top left-hand corner of μ_d. If $b = 0$ the block already has the desired format. If c is equal to zero, we can use "swapping" $U_{1,3}$ and suitable "adding" $V^{\alpha}_{1,3}$ matrices to perform Euclid's algorithm on the non-zero entries in the first and third columns, resulting in a block of the desired format, with $\lambda_1 = gcd(a, b)$ and $\lambda_2 = 0$, and the same type of procedure can be used when $a = 0$.

In the remaining case, when all entries are non-zero, we have to distinguish two scenarios: first, if $aq = b$ then $V^{qT}_{3,1} \mu_d V^{q}_{3,1}$ has the top left-hand block in Darboux normal form with $\lambda_1 = a$ and $\lambda_2 = c$.

Otherwise, performing Euclid's algorithm as above will initially result in a block with non-zero entries away from the main diagonal:

$$\begin{matrix} 0 & gcd(a,b) & 0 & y \\ -gcd(a,b) & 0 & 0 & 0 \\ 0 & 0 & 0 & x \\ -y & 0 & -x & 0 \end{matrix}$$

We now make a slight modification to the procedure in Lemma 23 in order to bring this block matrix back to standard form: instead of applying Euclid's algorithm to reduce the entries on the last row and column, we use it to reduce the entries on the first row and column. The resulting matrix will be of the form

$$\begin{matrix} 0 & gcd(a,b,y) & 0 & 0 \\ -gcd(a,b,y) & 0 & y' & 0 \\ 0 & -y' & 0 & x' \\ 0 & 0 & -x' & 0 \end{matrix}$$

At this point we can repeat the steps outlined so far until we eventually bring the block to Darboux normal form. Since we started with the assumption that a is not a factor of b, the greatest common divisor of a, b and y must be strictly less than a in the divisibility order, so eventually the process will terminate. ◀

Discussion. There is an important caveat to this result. We have defined contextuality for $\mathsf{G}(\mu)$ in terms of $\mathsf{C}(\mu)$, which is defined relative to the set of generators $\mathfrak{X}$. When transforming μ to μ' in Darboux normal form, we will have $\mathsf{G}(\mu) \cong H(\mu) \cong H(\mu') \cong \mathsf{G}(\mu')$, but this transformation will not preserve the generators. In particular, the new generators corresponding to the transformed basis for μ' may correspond to words in the old generators which cannot be formed from commuting products. This means that a contextual word over $\mathsf{G}(\mu')$ may not correspond to one over $\mathsf{G}(\mu)$.

6 Unitary representation

Since $\mathsf{G}(\mu)$ is a finite group, it has unitary representations. Indeed, every linear representation is equivalent to a unitary one. We wish to have unitary representations which faithfully preserve the internal $\mathbb{Z}_d$-action.

We use the qudit Hilbert space $\mathscr{H}_d := \mathbb{C}^d$, with basis vectors $|k\rangle$ labelled by elements of $\mathbb{Z}_d$. The tensor product of n copies of this space, $\mathscr{H}_{n,d}$, has basis vectors $|\vec{k}\rangle$ labelled by $\vec{k} \in \mathbb{Z}_d^n$. We write $\mathsf{U}(\mathscr{H}_{n,d})$ for the unitary group on $\mathscr{H}_{n,d}$. The centre of this group is isomorphic to the circle group, $\mathsf{U}(1) := \{z \in \mathbb{C} \mid |z| = 1\}$. For each $d \geq 2$, this contains the cyclic subgroup of the d'th complex roots of unity. We write $\omega := e^{\frac{2\pi i}{d}}$ for the primitive d'th root of unity. The map $k \mapsto \omega^k$ is an isomorphism from $\mathbb{Z}_d$ to the multiplicative group of d'th complex roots of unity.

Given a commutator matrix $\mu \in \mathfrak{so}(n, \mathbb{Z}_d)$, we shall define a representation $\rho : H(\mu) \to \mathsf{U}(\mathscr{H}_{n,d})$: $\rho(k, \vec{k}) |\vec{l}\rangle \;=\; \omega^{k+\check{\mu}(\vec{k},\vec{l})} |\vec{l} + \vec{k}\rangle$.

▶ **Proposition 25.** *For each $(k, \vec{k}) \in H(\mu)$, $\rho(k, \vec{k})$ is a well-defined unitary operation. Moreover, ρ is an injective group homomorphism which preserves scalars, i.e. $\rho(k, 0) = \omega^k \mathbb{1}$.*

Proof. The verification that ρ is a homomorphism amounts to showing that

$$\rho(k, \vec{k}) \circ \rho(k', \vec{k}') |\vec{l}\rangle \;=\; \rho(k + k' + \check{\mu}(\vec{k}, \vec{k}'), \vec{k} + \vec{k}') |\vec{l}\rangle$$

which reduces to

$$\check{\mu}(\vec{k}, \vec{k}') + \check{\mu}(\vec{k} + \vec{k}', \vec{l}) \;=\; \check{\mu}(\vec{k}', \vec{l}) + \check{\mu}(\vec{k}, \vec{k}' + \vec{l})$$

which follows from bilinearity. ◀

Representation in Pauli groups

The generalized Pauli groups $\mathbb{P}_{n,d}$ are the subgroups of $\mathsf{U}(\mathscr{H}_{n,d})$ generated by the X and Z operations. These operations are defined on $\mathscr{H}_d$ by $X |k\rangle = |k+1\rangle$, and $Z |k\rangle = \omega^k |k\rangle$. These are the Sylvester "shift" and "clock" matrices [20], and can be seen as discrete versions of position and momentum operators. Note that they satisfy the basic commutation relation $ZX = \omega XZ$. They are then extended to $\mathscr{H}_{n,d}$ as $X_i := \underbrace{I \otimes \cdots \otimes I}_{i-1} \otimes X \otimes \underbrace{I \otimes \cdots \otimes I}_{n-i}$, and similarly for Z_i, $i = 1, \ldots, n$. Note that the commutator matrix for the generators X_i, Z_i is in Darboux normal form: the only non-zero entries are $\mu(Z_i, X_i) = 1$, $\mu(X_i, Z_i) = -1 \pmod d$.

▶ **Proposition 26.** *The image of $H(\mu)$ under ρ is a subgroup of $\mathbb{P}_{n,d}$.*

Proof. Given an element e_i of the standard basis of $\mathbb{Z}_d^n$, we have $\rho(e_i) = X_i \prod_{j=1}^n Z_j^{\check{\mu}_{i,j}}$, which can be verified by a simple computation:

$$\begin{aligned} X_i \prod_{j=1}^n Z_j^{\check{\mu}_{i,j}} |k_1\rangle \otimes \cdots \otimes |k_n\rangle &= X_i(\omega^{\check{\mu}_{i,1}k_1}|k_1\rangle \otimes \cdots \otimes \omega^{\check{\mu}_{i,n}k_n}|k_n\rangle) \\ &= X_i(\prod_j \omega^{\mu_{i,j}k_j})\,|\vec{k}\rangle \\ &= \omega^{\sum_j \check{\mu}_{i,j}k_j} X_i\,|\vec{k}\rangle \\ &= \omega^{\check{\mu}(e_i,\vec{k})}\,|\vec{k}+e_i\rangle \\ &= \rho(e_i)\,|\vec{k}\rangle. \end{aligned}$$

Since $H(\mu)$ is generated by the e_i and the scalar $(1,0)$, this yields the result. ◀

This result shows the universality of the Pauli operations for expressing discrete commutation relations. At the same time, the structural tools made available by the presentations of commutation groups allow for a fine-grained analysis of the "algebra of contextuality".

7 Outlook

Non-commutativity is a fundamental mathematical feature of quantum mechanics, distinguishing it from classical physics. But in many key cases, we do not simply have the failure of commutativity, but rather that commutativity holds *up to* a specified scalar. This is the phenomenon of *commutation relations*, which play a central role in quantum physics. There are many familiar examples.

In this paper, we have given an answer, in the discrete case working over $\mathbb{Z}_d$, to the question: what *is* a commutation relation in general? This opens up the possibility of classifying the possible contextual behaviours arising from commutation relations. By virtue of the existence of unitary representations, these arise within quantum mechanics.

We mention a few topics of current and future work:

- Studying the cohomology of commutation groups, and relating this to the cohomological criteria for contextuality studied e.g. in [5, 3, 16, 1].
- Studying commutation groups in relation to state-dependent contextuality and empirical models [4].
- Relating commutation groups to the logical analysis of contextuality in terms of partial Boolean algebras [13, 2].
- Generalizing commutation groups to more general abelian groups of scalars.
- A Stone-von Neumann type theorem for commutation groups.

References

1 Sivert Aasnæss. Comparing two cohomological obstructions for contextuality, and a generalised construction of quantum advantage with shallow circuits. *arXiv preprint*, 2022. arXiv:2212.09382.

2 Samson Abramsky and Rui Soares Barbosa. The logic of contextuality. In Christel Baier and Jean Goubault-Larrecq, editors, *29th EACSL Annual Conference on Computer Science Logic, CSL 2021, January 25-28, 2021, Ljubljana, Slovenia (Virtual Conference)*, volume 183 of *LIPIcs*, pages 5:1–5:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.CSL.2021.5.

3 Samson Abramsky, Rui Soares Barbosa, Kohei Kishida, Raymond Lal, and Shane Mansfield. Contextuality, cohomology and paradox. In Stephan Kreutzer, editor, *24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany*, volume 41 of *LIPIcs*, pages 211–228. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.CSL.2015.211.

4 Samson Abramsky and Adam Brandenburger. The sheaf-theoretic structure of non-locality and contextuality. *New Journal of Physics*, 13(11):113036, 2011.

5 Samson Abramsky, Shane Mansfield, and Rui Soares Barbosa. The cohomology of non-locality and contextuality. In Bart Jacobs, Peter Selinger, and Bas Spitters, editors, *Proceedings 8th International Workshop on Quantum Physics and Logic, QPL 2011, Nijmegen, Netherlands, October 27-29, 2011*, volume 95 of *EPTCS*, pages 1–14, 2011. doi:10.4204/EPTCS.95.1.

6 Hannes Bartosik, Jürgen Klepp, Claus Schmitzer, Stephan Sponar, Adán Cabello, Helmut Rauch, and Yuji Hasegawa. Experimental test of quantum contextuality in neutron interferometry. *Physical review letters*, 103(4):040403, 2009.

7 Sergey Bravyi, David Gosset, Robert Koenig, and Marco Tomamichel. Quantum advantage with noisy shallow circuits. *Nature Physics*, 16(10):1040–1045, 2020.

8 Sergey Bravyi, David Gosset, and Robert König. Quantum advantage with shallow circuits. *Science*, 362(6412):308–311, 2018.

9 Richard Cleve, Li Liu, and William Slofstra. Perfect commuting-operator strategies for linear system games. *Journal of Mathematical Physics*, 58(1), 2017.

10 Yuji Hasegawa, Rudolf Loidl, Gerald Badurek, Matthias Baron, and Helmut Rauch. Quantum contextuality in a single-neutron optical experiment. *Physical Review Letters*, 97(23):230401, 2006.

11 Thomas F Jordan. *Quantum mechanics in simple matrix form*. Wiley, 1986.

12 William M Kirby and Peter J Love. Variational quantum eigensolvers for sparse Hamiltonians. *Physical Review Letters*, 127(11):110503, 2021.

13 Simon Kochen and Ernst P. Specker. The problem of hidden variables in quantum mechanics. *Journal of Mathematics and Mechanics*, 17(1):59–87, 1967.

14 N David Mermin. Simple unified form for the major no-hidden-variables theorems. *Physical review letters*, 65(27):3373, 1990.

15 N David Mermin and Rüdiger Schack. Homer nodded: von Neumann's surprising oversight. *Foundations of Physics*, 48:1007–1020, 2018.

16 Cihan Okay, Sam Roberts, Stephen D Bartlett, and Robert Raussendorf. Topological proofs of contextuality in quantum mechanics. *arXiv preprint*, 2017. arXiv:1701.01888.

17 Robert Raussendorf. Contextuality in measurement-based quantum computation. *Physical Review A*, 88(2):022322, 2013.

18 Stephen Semmes. An introduction to Heisenberg groups in analysis and geometry. *Notices of the AMS*, 50(6):640–646, 2003.

19 William Slofstra. Tsirelson's problem and an embedding theorem for groups arising from non-local games. *Journal of the American Mathematical Society*, 33(1):1–56, 2020.

20 James Joseph Sylvester. *On quaternions, nonions, sedenions, etc.* Johns Hopkins, 1883.

A The Pauli group on qubits

We recall the definition of the **Pauli operators**, dichotomic (*i.e.* two-valued) observables corresponding to measuring spin in the x, y, and z axes, with eigenvalues ± 1

$$X \coloneqq \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad Y \coloneqq \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad Z \coloneqq \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

These matrices are self-adjoint, have eigenvalues ± 1, and together with the identity matrix I satisfy the following relations:

$$\begin{aligned} &X^2 = Y^2 = Z^2 = I \\ &XY = iZ, \quad YZ = iX, \quad ZX = iY, \\ &YX = -iZ, \quad ZY = -iX, \quad XZ = -iY. \end{aligned} \tag{7}$$

Böhm and Taylor for All!

Aloÿs Dufour
Université Sorbonne Paris Nord, LIPN, CNRS, Villetaneuse, France

Damiano Mazza
CNRS, LIPN, Université Sorbonne Paris Nord, Villetaneuse, France

Abstract

Böhm approximations, used in the definition of Böhm trees, are a staple of the semantics of the lambda-calculus. Introduced more recently by Ehrhard and Regnier, Taylor approximations provide a quantitative account of the behavior of programs and are well-known to be connected to intersection types. The key relation between these two notions of approximations is a commutation theorem, roughly stating that Taylor approximations of Böhm trees are the same as Böhm trees of Taylor approximations. Böhm and Taylor approximations are available for several variants or extensions of the lambda-calculus and, in some cases, commutation theorems are known. In this paper, we define Böhm and Taylor approximations and prove the commutation theorem in a very general setting. We also introduce (non-idempotent) intersection types at this level of generality. From this, we show how the commutation theorem and intersection types may be applied to any calculus embedding in a sufficiently nice way into our general calculus. All known Böhm-Taylor commutation theorems, as well as new ones, follow by this uniform construction.

2012 ACM Subject Classification Theory of computation → Linear logic; Theory of computation → Denotational semantics; Theory of computation → Operational semantics; Theory of computation → Lambda calculus; Theory of computation → Process calculi

Keywords and phrases Linear logic, Differential linear logic, Taylor expansion of lambda-terms, Böhm trees, Process calculi

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.29

Acknowledgements The authors wish to thank Michele Pagani, who pointed out Lemma 23.

1 Introduction

Böhm trees [4, 5] and their variants are a staple of the semantics of λ-calculus. Intuitively, they are possibly infinite normal forms representing the essence of the behavior of a λ-term. As such, they may be seen as limits of a set of finite approximations, called *Böhm approximations*, describing larger and larger portions of the Böhm tree.

More recently, Ehrhard and Regnier introduced another notion of approximation for the λ-calculus, underlying their *Taylor expansion* [23, 25]. Based on the idea that programs may be seen as analytic functions on certain topological vector spaces [16, 17], this notion may be stripped of its quantitative aspects (the coefficients of the Taylor series) and, like a Böhm tree, be presented as a set of finite approximations. These *Taylor approximations* are syntactically very different from Böhm approximations: they are not necessarily normal forms, and they are linear, in the sense of linear logic [27] (they are related to Boudol's *terms with multiplicities* [7]), so their execution length is bounded by their size.

Given a λ-term t, since a Böhm approximation is, essentially, also a λ-term, one may take the set of all Taylor approximations of all Böhm approximations of t, yielding a set $\mathcal{T}(\mathrm{BT}(t))$. On the other hand, since Taylor approximations always normalize, one may take the set of all normal forms of all Taylor approximations of t, yielding a set $\mathrm{NF}(\mathcal{T}(t))$. A key result of Ehrhard and Regnier, known as the *commutation theorem*, states that $\mathcal{T}(\mathrm{BT}(t)) = \mathrm{NF}(\mathcal{T}(t))$. This connection between Böhm and Taylor approximations is a surprisingly powerful tool, implying a wealth of fundamental theorems in the pure λ-calculus [3].

The theory of Taylor approximations has been extended to call-by-value [18], to call-by-push-value [10] and to the $\lambda\mu$-calculus [2]. In the call-by-value setting, where a notion of Böhm tree is available, the commutation theorem is known to hold [29]. These results exploit the fact that, in reality, the notion of Taylor expansion exists in the much more general setting of *differential linear logic* [20]. Informally, whenever a system $\mathcal{S}$ may be encoded in differential linear logic, a notion of Taylor approximation for $\mathcal{S}$ may be "pulled back" along the encoding.

Interestingly, usual Böhm trees, as well as call-by-value Böhm trees, may also be seen as resulting from a general notion of "Böhm tree" for linear logic, pulled back along Girard's call-by-name and call-by-value encodings. It is natural to wonder whether the different forms of the commutation theorem are also "pullbacks" of a more general commutation theorem.

The main goal of this paper is to give an affirmative answer to the above question. We introduce a process calculus $\mathcal{P}roc$ corresponding to a form of differential linear logic, define Böhm and Taylor approximations for it and prove the commutation theorem at this level of generality. We also show that, as soon as a system $\mathcal{S}$ embeds in $\mathcal{P}roc$ in a sufficiently nice way, then Böhm trees, Taylor expansion and the commutation theorem automatically "pull back" to $\mathcal{S}$. The known commutation theorems of [23, 25, 29] are covered by our results. Furthermore, we show how Böhm trees for call-by-push-value and a fragment of the π-calculus may also be defined.

It is well-known that intersection types [11, 6], in particular in their non-idempotent version [26, 15], are strongly related to Taylor approximations [14, 18, 32]. We provide a system of non-idempotent intersection types for $\mathcal{P}roc$ and show how it too "pulls back" along embeddings, automatically characterizing the existence of normal forms via typability, as long as these are properly reflected in $\mathcal{P}roc$. Known examples, such as the above or those of [8], fit in this picture. From [32], we know that other forms of intersection types (affine, idempotent) may be treated in a similar way, but in this paper we restrict to the linear non-idempotent case for briefness.

For the reader acquainted with proof nets, our calculus $\mathcal{P}roc$ is a process calculus representation of untyped *proof structures* (*i.e*, not necessarily logically correct objects) of differential linear logic. More precisely, of a non-polarized version of Honda and Laurent proof structures [28]. In §5.1 we give a more precise statement about the connection with linear logic proofs. The reduction rules of $\mathcal{P}roc$ reflect the process calculi tradition and are less fine-grained than those of usual differential nets [24, 34, 33], while still being semantically correct. We do not claim any canonicity of $\mathcal{P}roc$: any other syntax for classical differential linear logic (for instance, an extension of Accattoli's syntax [1]) would probably work. The only real merit of $\mathcal{P}roc$ is to provide a manageable syntax.

Another difference is that we do not consider formal sums. In the literature on differential λ-calculi, formal sums as used to represent non-determinism: a non deterministic choice like $t \to u_1$ and $t \to u_2$ is expressed by the deterministic reduction $t \to u_1 + u_2$. By contrast, as customary in process calculi, there are no formal sums in $\mathcal{P}roc$ and we allow uncontrolled non-determinism (the calculus does not enjoy confluence). Also, our Taylor approximations are *rigid* in the sense of [35, 32]. Intuitively, this is implicit in the usual communication rule of polyadic process calculi, which is something like $\overline{a}\langle b_1, b_2\rangle \mid a(c_1, c_2).P \to P\{b_1/c_1\}\{b_2/c_2\}$ (cf. Definition 7, rule $\otimes/\parr$). Non-rigid syntaxes like the ones traditionally considered for defining Taylor approximations would correspond to b_1, b_2 being unordered in the output $\overline{a}\langle b_1, b_2\rangle$ (*i.e*, they form a multiset rather than a list) and the above reduction would become $\overline{a}\langle b_1, b_2\rangle \mid a(c_1, c_2).P \to P\{b_1/c_1\}\{b_2/c_2\} + P\{b_2/c_1\}\{b_1/c_2\}$, that is, we consider every possible ordering of the multiset and use formal sums to collect all results. Using rigidity and

dispensing with sums yields great syntactic simplifications with essentially no semantic loss. We do, however, neglect quantitative aspects (the coefficients of the Taylor series). Luckily, in a great number of interesting cases (for instance those of [3]), these are not needed.

About the applicability of our work, we must stress that the fact that an embedding $\mathcal{S} \to \mathcal{P}roc$ immediately establishes a nice theory of Böhm and Taylor approximations for $\mathcal{S}$ does not say anything about the actual interest of this theory. First of all, as is, the theory is formulated in the syntax of $\mathcal{P}roc$, and reformulating it in "$\mathcal{S}$-friendly" syntax is not automatic (the call-by-push-value case of §6.1 is an example). Second, its relevance and usefulness must be ascertained case by case. For example, we have not yet investigated the Böhm trees for the stack calculus or for the fragment of the π-calculus presented in §6.2 and §6.3.

Finally, we want to point out that our notion of embedding (Definition 1) is rather naive. While sufficient for the examples shown in this paper, many encodings, especially of process calculi, are not embeddings in the technical sense of this paper, typically because they are up to some equivalence. For instance, the encoding of the $\lambda\mu$-calculus in the stack calculus mentioned in §6.2 is up to β-equivalence, so it is not directly covered by this paper. Extending our results of §4.3 and §5.2 to a more general class of embeddings is an interesting subject for future work.

2 Preliminaries

In this paper we consider *reduction systems*, which are given by:

- a set of *objects*, to be thought of as terms, proofs, programs, states...
- a binary relation $\to^*$ which is reflexive and transitive.

So a reduction system is exactly the same thing as a preorder.

A *morphism* of reduction systems $f : \mathcal{S} \to \mathcal{T}$ is just a monotonic function: $s \to^* s'$ implies $f(s) \to^* f(s')$.

▶ **Definition 1** (embedding). *An* embedding $f : \mathcal{S} \to \mathcal{T}$ *is a morphism such that, for every object s of $\mathcal{S}$, $f(s) \to^* t'$ implies that there exists s' such that $t' \to^* f(s')$ and $s \to^* s'$.*

The following is straightforward:

▶ **Lemma 2.** *If $f : \mathcal{S} \to \mathcal{T}$ and $g : \mathcal{T} \to \mathcal{U}$ are embeddings, then $g \circ f$ is an embedding.*

3 Processes

We fix two disjoint, countably infinite sets of *linear names*, ranged over by $a, b, c \dots$ and *cartesian names*, ranged over by $x, y, z \dots$ As customary in process calculi, we denote by $\widetilde{a}$ sequences (possibly empty) of linear names, and we write $|\widetilde{a}|$ for the length of $\widetilde{a}$.

▶ **Definition 3** (pre-process, context). Pre-processes *are defined as follows:*

$$P, Q ::= \mathbf{0} \mid P \mid Q \mid a \leftrightarrow b \mid \overline{a}\langle\widetilde{b}\rangle \mid a\langle\widetilde{b}\rangle \qquad \textit{(linear pre-processes)}$$
$$\mid \nu x P \mid \overline{a}(x)P \mid a(x)P \mid \overline{x}\langle a\rangle \mid x\langle a\rangle \mid !x(a).P$$

Linear pre-processes *are those generated only by the first line. We use $p, q \dots$ to range over linear pre-processes. Pre-processes of the form $!x(a).P$ are called* boxes. *In the literature on process calculi, $a \leftrightarrow b$ is usually called* linear forwarder*; we will call it* axiom *to stress the connection with proof nets.*

Contexts *are defined as pre-processes but with the addition of the* hole $\{-\}$*. As customary, we consider only contexts having exactly one occurrence of the hole, and denote them by* C*. We denote by* $\mathsf{C}\{P\}$ *the pre-process obtained by plugging pre-process* P *in the hole of* C*.*

▶ **Definition 4** (occurrences of names)**.** *We say that a linear name* a *occurs as* subject *in* $\overline{a}\langle\widetilde{b}\rangle$*,* $a\langle\widetilde{b}\rangle$*,* $\overline{a}(x)P$*,* $a(x)P$*,* $a \leftrightarrow b$ *and* $b \leftrightarrow a$*. All other occurrence of* a *are as* object*.*

The notations νx*,* $\overline{a}(x)$ *and* $a(x)$ *are called* cartesian binders*: in* νxP*,* $\overline{a}(x)P$ *and* $a(x)P$ *the cartesian name* x *is bound, and* α*-equivalence applies as usual. A cartesian name which is not bound is* free*.*

If a cartesian name x *appears elsewhere than in a binder, then we say that it appears as* subject*, and such an occurrence is* positive *if it is of the form* $\overline{x}\langle a\rangle$*, or* negative *if it is of the form* $x\langle a\rangle$ *or* $!x(a).P$*.*

▶ **Definition 5** (process)**.** *A (linear)* process *is a (linear) pre-process verifying the following:*

- *every linear name occurs at most twice. If it occurs once, we say that it is* free*, otherwise it is* bound *and* α*-equivalence applies to it. We write* $\mathsf{fn}(P)$ *for the set of free names of* P*, both linear and cartesian.*
- *In* $\overline{a}(x)P$ *(resp.* $a(x)P$*) every free occurrence of* x *in* P *(if any) is positive (resp. negative).*
- *In* $!x(a).P$*,* $\mathsf{fn}(P) = \{a\} \cup X$ *where* X *consists solely of cartesian variables with positive occurrences in* P *(the case* $X = \emptyset$ *is allowed).*

▶ **Definition 6** (structural congruence)**.** *Structural congruence is the relfexive, symmetric, transitive and contextual closure of the following rules:*

$$(P \mid Q) \mid R \equiv P \mid (Q \mid R) \qquad P \mid \mathbf{0} \equiv P \qquad P \mid Q \equiv Q \mid P \qquad \nu x\mathbf{0} \equiv \mathbf{0}$$

$$\beta P \mid Q \equiv \beta(P \mid Q) \qquad \beta\gamma P \equiv \gamma\beta P \qquad a \leftrightarrow b \equiv b \leftrightarrow a \qquad a \leftrightarrow b \mid P \equiv P\{a/b\},$$

where: in the bottom row, β *and* γ *are cartesian binders and* β *binds* $x \notin \mathsf{fn}(Q)$*; in the last equation, we require* $b \in \mathsf{fn}(P)$*, and* $P\{a/b\}$ *means, as customary, that we rename the unique free occurrence of* b *in* P *with* a*.*

One may readily check that being a process is preserved under structural congruence.

For the reader acquainted with proof nets of differential linear logic, the following "dictionary" may be helpful (the unacquainted reader may safely ignore this paragraph): $\mathbf{0}$ is the empty net; $P \mid Q$ is juxtaposition of two nets; as mentioned above, $a \leftrightarrow b$ is an axiom; $\overline{a}\langle\widetilde{b}\rangle$ and $a\langle\widetilde{b}\rangle$ are n-ary tensor and par nodes, respectively; νxP is an "exponential cut" ("linear cuts" are represented as explained in Definition 8 below); $\overline{a}(x)P$ and $a(x)P$ are n-ary contraction and cocontraction, respectively (the arity is the number of occurrences of x in P); $\overline{x}\langle a\rangle$ and $x\langle a\rangle$ are derelection and coderelection nodes, respectively; as already mentioned, $!x(a).P$ is an exponential box, with principal port x and as many auxiliary ports as there are occurrences of free names in P. The idea is that the free names of a process correspond to the conclusions of an untyped net. Structural congruence corresponds to equality of nets (in the sense of graphical representations), plus elimination of cuts with axioms (this is consistent with interaction nets [24], in which axioms are just wires).

▶ **Definition 7** (reduction)**.** *The basic reduction rules are follows:*

$$\begin{array}{rlll}
\overline{a}\langle\widetilde{b}\rangle \mid a\langle\widetilde{c}\rangle \mid P & \to_{\otimes/\parr} & P\{\widetilde{b}/\widetilde{c}\} & |\widetilde{b}| = |\widetilde{c}|,\ \widetilde{c} \in \mathsf{fn}(P) \\
\overline{a}(x)P \mid a(x)Q & \to_{!/?} & \nu x(P \mid Q) & \\
\nu x(\overline{x}\langle a\rangle \mid x\langle b\rangle \mid P) & \to_{\mathsf{cod}_0} & \nu xP\{a/b\} & b \in \mathsf{fn}(P) \\
\nu x(\mathsf{C}\{\overline{x}\langle a\rangle\} \mid !x(b).P) & \to_{\mathsf{c}} & \nu x(\mathsf{C}\{P\{a/b\}\} \mid !x(b).P) & \\
\nu x(!x(a).P \mid Q) & \to_{\mathsf{w}} & \nu xQ & x \notin \mathsf{fn}_+(Q)
\end{array}$$

plus the rule

$$\nu x(!y(c).\mathsf{C}\{\overline{x}\langle a\rangle\} \mid x\langle b\rangle \mid P) \;\to_{\mathsf{cod}_!}\; \nu x(!y(c).\nu w\mathsf{C}\{\overline{w}\langle a\rangle\} \mid y\langle c\rangle \mid \nu z(\mathsf{C}\{\overline{z}\langle a\rangle\} \mid z\langle b\rangle) \mid P)$$

in which w and z are fresh names. In the w *rule, $x \notin \mathsf{fn}_+(Q)$ means that x has no positive free occurrence in Q.*

If $R \to R'$ by means of one of the above basic reductions, we write $P \to Q$ to say there exists a context C such that $P \equiv \mathsf{C}\{R\}$ and $Q \equiv \mathsf{C}\{R'\}$. We write $\to^*$ for the reflexive-transitive closure of $\to$. This induces a reduction system $\mathcal{P}roc$ (resp. $\mathcal{L}in\mathcal{P}roc$) whose objects are processes (resp. linear processes) modulo structural equivalence.

▶ **Definition 8** (cut-free process). *A* linear cut *in a process is a linear name occurring twice as subject. A* cartesian cut *is any subprocess of the form $\nu x P$. A process is* cut-free *if it contains neither linear nor cartesian cuts. A process P* has a cut-free form *if $P \to^* N$ with N cut-free.*

The next two paragraphs provide additional explanations for linear logic experts, which unacquainted readers may ignore. The basic rules of Definition 7 are a reformulation of cut-elimination steps of differential linear logic proof nets. The rule $\otimes/\parr$ is the multiplicative step. The rule !/? reduces a cut between a contraction and a cocontraction. However, instead of the usual rule making the two commute with each other, this rule creates a cartesian cut (which we could also call "exponential cut"). Cartesian cuts should be seen as *communication zones* in the sense of Ehrhard and Laurent [22]: any number of derelictions and coderelections/boxes (as many as the premises of the contraction and the cocontraction originating the cut) may be paired up non-deterministically in a cartesian cut, using the rules cod_0 (derelection/codereliction), $\mathsf{cod}_!$ (a codereliction interacting with the border of a box) or c (a dereliction extracting one copy of a box). The rule w erases a box when there is no dereliction left. Intuitively, we are taking communication zones as primitive, rather than implementing them as Ehrhard and Laurent do in [22].

In fact, the usual presentation of cut-elimination of differential linear logic [24, 34, 33] is based on finer-grained rules than ours. In their encoding of process calculi, Ehrhard and Laurent use this finer granularity to implement "communication zones" ensuring that inputs may interact with outputs [22]. This means, in particular, that our formulation is semantically correct with respect to the usual one: the finer-granularity rules may simulate our rules. However, our formulation has the advantage of matching the tradition of process calculi, as well as solving the problems pointed out in [31], which are precisely due to the fact that the usual cut-elimination rules are too fine-grained for expressing concurrency in the same way as process calculi.

Nevertheless, $\mathcal{P}roc$ does exhibit some unusual features with respect to standard process calculi. The most striking one is perhaps the presence of linear names and the convention that a linear name is bound as soon as it appears twice. For instance, the process $\overline{a}\langle\rangle \mid a\langle\rangle \mid \overline{b}\langle\rangle$ would be traditionally written $\nu a(\overline{a}\langle\rangle \mid a\langle\rangle \mid \overline{b}\langle\rangle)$. This would lead to a proliferation of ν's, so we chose to leave these name restrictions implicit.

Another remarkable difference is that we allow reduction to occur inside boxes, *i.e*, if $P \to Q$, then $!x(a).P \to !x(a).Q$. This is useful for encoding λ-calculi whose reductions may occur in arbitrary positions. It is essential for recovering usual Böhm trees, for example. In §5.2 we will consider *shallow reduction*, which only happens outside of boxes and corresponds to standard reduction of process calculi. This is the notion of reduction to which intersection types are most immediately applicable. It is also useful for encoding weak reduction strategies of λ-calculi, which do not reduce under λ.

Observe that all basic reductions involve a cut. Therefore, a cut-free process is normal with respect to reduction. The converse is false: for example, $\overline{a}\langle\rangle \mid a\langle b\rangle$ is a normal process containing a cut with an "arity mismatch". Cuts to which no reduction applies are called *irreducible*. Other examples of irreducible linear cuts are "clashes" such as $\overline{a}\langle b\rangle \mid a(x)P$ or $\overline{a}\langle b\rangle \mid \overline{a}\langle c\rangle$, or "vicious circles" such as $a \leftrightarrow a$ (for the acquainted reader: the first corresponds to a cut beween a tensor and a contraction; the second to a cut between two tensors; the third to a cut between the two conclusions of an axiom). Irreducible cartesian cuts are of the form $\nu x(\overline{x}\langle a\rangle \mid P)$ (resp. $\nu x(x\langle a\rangle \mid P)$) with P containing no negative (resp. positive) occurrence of x (for the acquainted reader: the first corresponds to a cut between a dereliction and a coweakening, the second to a cut between a codereliction and a weakening). In §5.1 we will introduce types and correctness. The former eliminate clashes and arity mismatches, the latter vicious circles. Irreducible cartesian cuts, instead, may be present even in well-typed correct processes. They correspond to situations which, in the usual syntax of differential linear logic using formal sums, reduce to the empty sum.

The following lemma will be useful later:

▶ **Lemma 9.** *$P \to^* Q$ implies $P\{a/b\} \to^* Q\{a/b\}$.*

Proof. By induction on the length of the reduction, we reduce to the case of a single step $\mathsf{C}\{R\} \to \mathsf{C}\{R'\}$, which is proved by induction on C and by inspection of Definition 7. ◀

4 Approximations

4.1 Taylor Approximations

▶ **Definition 10** (Taylor process)**.** Taylor processes *are linear processes in which certain inputs and outputs are marked as "special" and denoted by $\overline{a}\langle\!\langle\widetilde{b}\rangle\!\rangle$ and $a\langle\!\langle\widetilde{b}\rangle\!\rangle$. Reduction is defined just as in linear processes, i.e, with rule $\otimes/\parr$ of Definition 7, but restricted to special/special and non-special/non-special pairs, that is, $\overline{a}\langle\widetilde{b}\rangle \mid a\langle\!\langle\widetilde{c}\rangle\!\rangle$ is irreducible even in case $|\widetilde{b}| = |\widetilde{c}|$. We denote by $\mathcal{T}ay$ the reduction system with Taylor processes as objects.*

There is an obvious morphism $\mathcal{T}ay \to \mathcal{L}in\mathcal{P}roc$ which forgets the "special" annotations. It is *not* an embedding: $\overline{a}\langle\!\langle\rangle\!\rangle \mid a\langle\rangle$ is mapped to $\overline{a}\langle\rangle \mid a\langle\rangle$, which reduces to $\mathbf{0}$, but the original process cannot reduce.

▶ **Lemma 11.** *In Taylor processes, reduction is strongly confluent and terminating.*

Proof. The only reduction steps are of kind $\otimes/\parr$, which cannot overlap, hence strong confluence. For what concerns termination, define the size of a Taylor process to be the number of axioms and subprocesses of the form $\overline{a}\langle\widetilde{b}\rangle$, $a\langle\widetilde{b}\rangle$, $\overline{a}\langle\!\langle\widetilde{b}\rangle\!\rangle$, $a\langle\!\langle\widetilde{b}\rangle\!\rangle$ which are present in the process. By inspecting Definition 6 and the $\otimes/\parr$ rule, we immediately see that size is preserved by structural congruence and that it strictly decreases under reduction, which implies termination. ◀

The *Taylor approximation relation* is defined in Fig. 1. It uses *approximation judgments* of the form $p \sqsubset P \vdash \Xi; \Xi'$ where:

- p is a Taylor process and P an arbitrary process;
- Ξ and Ξ' are disjoint finite sets of pairs of the form $a \sqsubset x$, where a is a linear name not appearing free in P and x a cartesian name, such that every linear name appears at most once in $\Xi \cup \Xi'$.

$$\frac{}{\mathbf{0} \sqsubset \mathbf{0} \vdash ;} \qquad \frac{p \sqsubset P \vdash \Xi ; \Xi' \quad q \sqsubset Q \vdash \Upsilon ; \Upsilon'}{p \mid q \sqsubset P \mid Q \vdash \Xi, \Upsilon ; \Xi', \Upsilon'} \qquad \frac{}{a \leftrightarrow b \sqsubset a \leftrightarrow b \vdash ;}$$

$$\frac{}{\overline{a}\langle \widetilde{b} \rangle \sqsubset \overline{a}\langle \widetilde{b} \rangle \vdash ;} \qquad \frac{}{a\langle \widetilde{b} \rangle \sqsubset a\langle \widetilde{b} \rangle \vdash ;} \qquad \frac{p \sqsubset P \vdash \Xi, \widetilde{a} \sqsubset x ; \Xi', \widetilde{b} \sqsubset x}{p\{\widetilde{a}/\widetilde{b}\} \sqsubset \nu x P \vdash \Xi ; \Xi'} \; |\widetilde{a}| = |\widetilde{b}|$$

$$\frac{p \sqsubset P \vdash \Xi, \widetilde{b} \sqsubset x ; \Xi'}{\overline{a}\langle\!\langle \widetilde{b} \rangle\!\rangle \mid p \sqsubset \overline{a}(x) P \vdash \Xi ; \Xi'} \; x \notin \Xi \qquad \frac{p \sqsubset P \vdash \Xi ; \Xi', \widetilde{b} \sqsubset x}{a\langle\!\langle \widetilde{b} \rangle\!\rangle \mid p \sqsubset a(x) P \vdash \Xi ; \Xi'} \; x \notin \Xi'$$

$$\frac{}{a \leftrightarrow b \sqsubset \overline{x}\langle b \rangle \vdash a \sqsubset x ;} \; a \neq b \qquad \frac{}{a \leftrightarrow b \sqsubset x\langle b \rangle \vdash ; a \sqsubset x} \; a \neq b$$

$$\frac{p_1 \sqsubset P\{a_1/a\} \vdash \Xi_1 ; \quad \dots \quad p_n \sqsubset P\{a_n/a\} \vdash \Xi_n ;}{p_1 \mid \cdots \mid p_n \sqsubset !x(a).P \vdash \Xi_1, \dots, \Xi_n ; a_1 \sqsubset x, \dots, a_n \sqsubset x} \; \forall i \, a_i \notin \mathsf{fn}(P)$$

Figure 1 Taylor approximations.

The intuition is that $a \sqsubset x$ in Ξ (resp. Ξ') means that the linear name a approximates one positive (resp. negative) occurrence of the cartesian name x.

In the sequel, we will write $p \sqsubset P$ when a judgment of the form $p \sqsubset P \vdash \Xi ; \Xi'$ is derivable for some Ξ, Ξ'.

▶ **Lemma 12.** *Let $p \sqsubset P$. Then:*

1. *$P = Q\{a/b\}$ iff $p = q\{a/b\}$ for some $q \sqsubset Q$;*
2. *$P = \mathsf{C}\{Q\}$ with $\mathsf{fn}(Q) = \{a\} \cup X$ with X consisting of cartesian names with only positive occurrences implies $p \equiv t \mid q_1 \mid \cdots \mid q_n$ for some q_i and t such that $t \sqsubset \mathsf{C}\{\overline{x}\langle a \rangle\}$ whenever $x \in \mathsf{fn}(\mathsf{C}\{\overline{x}\langle a \rangle\})$, and $q_i \sqsubset Q\{a_i/a\}$ for all $1 \leq i \leq n$ and some pairwise distinct a_i;*
3. *$P \equiv P'$ (resp. $p \equiv p'$) implies $p \equiv p'$ for some p' (resp. $P \equiv P'$ for some P') such that $p' \sqsubset P'$.*

Proof. Point (1) is proved by induction on P. Point (2) is proved by induction on C. Point (3) is proved by checking every rule of Definition 6 and then by induction on contexts. ◀

Point (3) of Lemma 12 assures us that we may transparently use structural congruence with Taylor approximations, which is what we will do from now on.

We now prove the two fundamental properties of Taylor approximations, namely that they may be pulled back along arbitrary reductions and pushed forward along reductions to cut-free forms.

▶ **Lemma 13** (pull-back). *Let $P \to^* Q$ and $q \sqsubset Q$. Then, there exists $p \sqsubset P$ such that $p \to^* q$. Diagrammatically:*

$$\begin{array}{ccc} P & \longrightarrow^* & Q \\ & & \sqcup \\ & & q \end{array} \qquad \Longrightarrow \qquad \begin{array}{ccc} P & \longrightarrow^* & Q \\ \sqcup & & \sqcup \\ p & \longrightarrow^* & q \end{array}$$

Proof. We start by proving the lemma when $P = R$, $Q = R'$ and $R \to R'$ by means of one of the basic rules of Definition 7. The cases $\otimes/\parr$ and !/? are immediate applications of point (1) of Lemma 12. For the cases cod_0, $\mathsf{cod}_!$, c and w, one shows that $q \sqsubset R'$ implies $q \sqsubset R$, so the pull-back is the empty reduction. This claim is straightforward for the rules cod_0, $\mathsf{cod}_!$ and w; for the rule c, point (2) of Lemma 12 is used.

Next, we prove the lemma for one-step reduction, that is, when $P \equiv \mathsf{C}\{R\}$, $Q \equiv \mathsf{C}\{R'\}$ and $R \to R'$ by means of a basic rule. Define the *depth* of the context C to be number of nested boxes within which the hole is located. The proof is by induction on the depth of C. If the depth is zero, then $\mathsf{C} \equiv \nu\tilde{z}(S \mid \{-\})$ for some S, from which we infer $q \equiv (s \mid r')\sigma$ where $s \sqsubset S$, $r' \sqsubset R'$ and σ is a substitution. We apply the result we proved above, and obtain $r \sqsubset R$ such that $r \to^* r'$. Then, if we let $p := (s \mid r)\sigma$, we have $p \sqsubset P$ such that, using Lemma 9, $p \to^* q$ as desired. If C has depth $d+1$, then $\mathsf{C} \equiv \nu\tilde{z}(S \mid !x(a).\mathsf{C}')$ for some process S and context C' of depth d. We therefore have $q \equiv (s \mid q_1 \mid \cdots \mid q_n)\sigma$ with $s \sqsubset S$ and $q_i \sqsubset \mathsf{C}'\{R'\}\{a_i/a\}$ for all $1 \leq i \leq n$. By hypothesis and Lemma 9, we have $\mathsf{C}'\{R\}\{a_i/a\} \to \mathsf{C}'\{R'\}\{a_i/a\}$, so we apply the induction hypothesis to each q_i and obtain $p_i \sqsubset \mathsf{C}'\{R\}\{a_i/a\}$ such that $p_i \to^* q_i$. Then, if we let $p := (s \mid p_1 \mid \cdots \mid p_n)\sigma$, we have $p \sqsubset P$ such that $p \to^* q$, and we conclude.

Finally, we prove the general version of the lemma by induction on the length of the reduction $P \to^* Q$. The statement is trivial for length zero because $P \equiv Q$. For length $k+1$, we have $P \to P_1 \to^* Q$ with $P_1 \to^* Q$ of length k. Given $q \sqsubset Q$, the induction hypothesis gives us $p_1 \sqsubset P_1$ such that $p_1 \to^* q$. Then, we apply the one-step case proved above to p_1 and obtain the desired $p \sqsubset P$ such that $p \to^* p_1 \to^* q$. ◀

▶ **Lemma 14.** *Let $p \sqsubset P$ such that p has a cut-free form and $p \to p'$ (one-step reduction). Then, there exist reductions $p' \to^* q$ and $P \to Q$ such that $q \sqsubset Q$. Diagrammatically:*

$$\begin{array}{ccc} P & & P \longrightarrow Q \\ \sqcup & \Longrightarrow & \sqcup \quad\quad \sqcup \\ p \longrightarrow p' & & p \longrightarrow p' \longrightarrow^* q \end{array}$$

Proof. The proof is by induction on the structure of P. We only include the complicated cases, which are when P is a parallel composition, a box or a restriction. If $P = P_1 \mid P_2$, then we know that $p = p_1 \mid p_2$ with $p_i \sqsubset P_i$. The reduction may be performed within one of the p_i, in which case we use the induction hypothesis to conclude, or it may be a reduction between an output in, say, p_1 and an input in p_2. By Fig. 1, this means that P_1 is of the form $\overline{a}\langle\widetilde{b}\rangle \mid P_1'$ or $\overline{a}(x)P_1'$, and P_2 of the dual form, so P may perform a matching reduction.

If $P = !x(a).P'$, then $p = p_1 \mid \cdots \mid p_n$ with $p_i \sqsubset P'\{a_i/a\}$. We claim that the step $p \to p'$ cannot happen because of a communication between two distinct p_i. If this were the case, then we would have a free name b occurring as input subject in some p_i. Now, we cannot have $b = a_i$, because a_i does not appear in any other p_j with $j \neq i$. Then, since boxes may have no free linear name, b must approximate a free cartesian name of P', but this is impossible, because free cartesian names of boxes may only appear as outputs, and approximations of outputs are still outputs. So $p_j \to p_j'$ for some j, and $p' = p_1 \mid \cdots \mid p_j' \mid \cdots \mid p_n$. Since approximations do not introduce cuts, the cut reduced in $p_j \to p_j'$ comes from a cut of P'. Such a cut induces a cut in *every* p_i, because these all approximate P'. But p has a cut-free form, so all these cuts are reducible, therefore $p_i \to p_i'$ for *all* i. The induction hypothesis gives us $p_i' \to^* q_i$ and $P' \to Q_i'$ such that $q_i \sqsubset Q_i'$. However, since it is the same cut of P' that is being reduced in each $P' \to Q_i'$, all Q_i' are actually equal to some Q', so $!x(a).P \to^* !x(a).Q'$ and $q_1 \mid \cdots \mid q_n \sqsubset !x(a).Q'$, as desired.

If $P = \nu x P_1$, then $p = p_1\{\widetilde{a}/\widetilde{b}\}$ with $p_1 \sqsubset P_1$ and $\widetilde{a}, \widetilde{b}$ approximate x as output and input, respectively. In case the reduction $p \to p'$ is already present in p_1, *i.e*, in case $p' = p_1'\{\widetilde{a}/\widetilde{b}\}$ with $p_1 \to p_1'$, the induction hypothesis allows us to conclude immediately. Otherwise, the reduction is made possible by the substitution of a_i to b_i for some i, which means that there is an occurrence of x as input matching an occurrence of x as output in P_1. Such occurrences are uniquely determined by the index i and induce a one-step reduction in P. More precisely, we have $P = \nu x \mathsf{C}\{R\}$ and $R \to_{\mathsf{x}} R'$ where x is one of cod_0, $\mathsf{cod}_!$ or c. At this point, the proof splits in two cases, according to the shape of C. If C is shallow (*i.e*, the hole is not under a box), then we conclude straightforwardly (we omit the details). In case the hole is under a box, we use the same argument given above (for the case $P = !x(a).P'$) to conclude that the step $p \to p'$ may be "completed" by reducing the other cuts in p approximating the cut corresponding to $R \to_{\mathsf{x}} R'$, yielding the desired push-forward. ◀

▶ **Lemma 15** (push-forward). *Let $p \sqsubset P$ and $p \to^* n$ such that n is cut-free. Then, there exists a reduction $P \to^* Q$ such that $n \sqsubset Q$. Diagrammatically:*

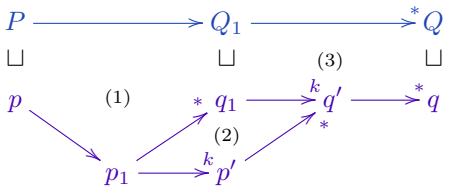
$$\begin{array}{ccc} P & & P \longrightarrow^* Q \\ \sqcup & \Longrightarrow & \sqcup \quad\ \sqcup \\ p \longrightarrow^* n & & p \longrightarrow^* n \end{array}$$

Proof. We prove the following stronger result, which is a generalization of Lemma 14 to reductions of arbitrary length: for all $p \sqsubset P$ with p having a cut-free form and for all $p \to^* p'$, there exist $p' \to^* q$ and $P \to^* Q$ such that $q \sqsubset Q$. Diagrammatically:

$$\begin{array}{ccc} P & & P \longrightarrow^* Q \\ \sqcup & \Longrightarrow & \sqcup \quad\ \sqcup \\ p \longrightarrow^* p' & & p \longrightarrow^* p' \longrightarrow^* q \end{array}$$

The lemma is the special case in which p' is cut-free, which implies $q = p'$.

The proof is by induction on the length of the reduction $p \to^* p'$. If the length is zero, then p is itself cut-free and the claim is immediate. Now suppose that the length is $k+1$, which means $p \to p_1 \to^k p'$, with $\to^k$ denoting reduction in k steps. Diagrammatically, the proof may be depicted as follows:



where (1) holds by Lemma 14, (2) by Lemma 11 (strong confluence) and (3) by the induction hypothesis. Let us spell out the details. By Lemma 14, there exist reductions $p_1 \to^* q_1$ and $P \to^* Q_1$ such that $q_1 \sqsubset Q_1$. Now, it is well known that, in a strongly confluent system, the two sides of confluence diamonds have the same length, that is, if $t \to^m t'$ and $t \to^n t''$, there there exists u such that $t' \to^n u$ and $t'' \to^m u$. Therefore, by Lemma 11, there exists q' such that $p' \to^* q'$ and $q_1 \to^k q'$. But q_1 still has a cut-free form, because it is a reduct of p, so the induction hypothesis allows us to conclude. ◀

Lemma 15 is why "special" inputs and outputs are needed in Taylor approximations. If, for instance, $\overline{a}(x)P$ where approximated by $\overline{a}\langle\widetilde{b}\rangle \mid p$ rather than $\overline{a}\langle\langle\widetilde{b}\rangle\rangle \mid p$, then the push-forward property would fail: for example, $\overline{a}(x) \mid a\langle\rangle$ cannot reduce, but would be approximated by $\overline{a}\langle\rangle \mid a\langle\rangle$, which reduces to $\mathbf{0}$. This is related to the failure of the forgetful map $\mathcal{T}ay \to \mathcal{L}in\mathcal{P}roc$ to be an embedding, as pointed out above.

4.2 Böhm Approximations and the Commutation Theorem

▶ **Definition 16** (Böhm approximations). *We define $P \leq_0 Q$ just if P is obtained from Q by replacing any number of its boxes (i.e, subprocesses of the form $!x(a).R$) with $\mathbf{0}$. The* Böhm approximation relation *is defined as follows: $N < P$ iff N is cut-free and there exists P' such that $P \to^* P'$ and $N \leq_0 P'$.*

Observe that Definition 16 applies to Taylor processes as well. However, in that case, $\leq_0$ degenerates to equality, because there are no boxes in Taylor processes, so $n < p$ simply means that n is the cut-free form of p (not all Taylor processes have one).

▶ **Lemma 17.** *For every cut-free Taylor process n, $n \sqsubset P$ iff there exists a cut-free process N such that $n \sqsubset N \leq_0 P$.*

Proof. Formally, both directions are by induction on P. Rather than detailing all cases, let us give the intuition. For the forward direction, the key point is that the only way Taylor approximations may "forget" cuts is by approximating a box with $\mathbf{0}$. So, if $n \sqsubset P$ with n cut-free, either P is already cut-free, and we are done, or all of its cuts are within boxes approximated by $\mathbf{0}$ in n. Therefore, we define N as P in which those same boxes are replaced with $\mathbf{0}$, and obtain N cut-free such that $n \sqsubset N \leq_0 P$.

The converse does not even rely on cut-freeness: we have $n \sqsubset N$ and N is obtained from P by replacing some boxes with $\mathbf{0}$, but $\mathbf{0}$ is a Taylor approximation of any box, so $n \sqsubset P$. ◀

▶ **Theorem 18** (Böhm-Taylor commutation). *The relations $\sqsubset<$ and $<\sqsubset$ coincide.*

Proof. The proof is in the following diagram:

$$\begin{array}{ccccc} P & \xrightarrow{*} & Q & \geq_0 & N \\ \sqcup & (1) & \sqcup & (2) & \sqcup \\ p & \xrightarrow{*} & n & = & n \end{array}$$

Indeed, observe that, by definition, $n \sqsubset< P$ is equivalent to the situation depicted in the red part of the diagram, for some Q and cut-free N. Similarly, $n <\sqsubset P$ is equivalent to the situation depicted in the green part of the diagram, for some p.

For red implies green, notice that Taylor approximations do not introduce cuts, so N cut-free implies n cut-free. Then, Lemma 17 gives us square (2), and Lemma 13 gives us square (1).

For green implies red, square (1) is given by Lemma 15 and square (2) by Lemma 17. ◀

4.3 Pulling Back the Commutation Theorem

In what follows, we fix an arbitrary reduction system $\mathcal{S}$ equipped with an embedding $f : \mathcal{S} \to \mathcal{P}roc$ (in the technical sense of Definition 1).

▶ **Definition 19** (Böhm tree). *Let s be an objects of $\mathcal{S}$. A* Böhm approximation *of s is a cut-free process N such that $s \to^* s'$ and $N \leq_0 f(s')$. The* Böhm tree *of s, denoted by* $\mathrm{BT}(s)$*, is the set of all Böhm approximations of s.*

Observe that we could have defined Böhm approximations by asking that $f(s) \to^* f(s')$ and $N \leq_0 f(s')$. We now prove that this is equivalent, because f is an embedding.

▶ **Lemma 20.** *For every process P and cut-free process N, $N \leq_0 P$ and $P \to^* P'$ implies $N \leq_0 P'$.*

Proof. Since N is cut-free, and since N differs from P only by the fact that some boxes of P are replaced by $\mathbf{0}$ in N, then any cut of P is inside a box. Therefore, P' differs from P only inside some boxes, and replacing them with $\mathbf{0}$ yields again N. ◀

▶ **Lemma 21.** *For every object s of $\mathcal{S}$, N is a Böhm approximation of s iff $f(s) \to^* f(s')$ such that $N \leq_0 f(s')$.*

Proof. The forward reduction is immediate: by definition, we have a reduction $s \to^* s'$ in $\mathcal{S}$ such that $N \leq_0 f(s')$, so we take the reduction $f(s) \to^* f(s')$ which we have by monotonicity. For the converse, suppose that $f(s) \to^* f(s')$. By definition of embedding, there exists s'' such that $f(s') \to^* f(s'')$ and $s \to^* s''$. But by Lemma 20, $N \leq_0 f(s')$ implies $N \leq_0 f(s'')$, so we conclude. ◀

The terminology "Böhm tree" is somewhat abusive because, strictly speaking, $\mathrm{BT}(s)$ has nothing of a tree. In good cases, however, it does have the properties of Böhm trees. First of all, it follows immediately from the definition that $s \to^* s'$ implies $\mathrm{BT}(s') \subseteq \mathrm{BT}(s)$. In case $\mathcal{S}$ is confluent, *i.e*, if any span of the form $s \to^* s_1$, $s \to^* s_2$ may be closed by reductions $s_1 \to^* s'$ and $s_2 \to^* s'$, then the converse implication also holds:

▶ **Proposition 22.** *Let $\mathcal{S}$ be confluent. Then, $s \to^* s'$ implies $\mathrm{BT}(s) = \mathrm{BT}(s')$.*

Proof. We only need to show that $\mathrm{BT}(s) \subseteq \mathrm{BT}(s')$. Let $N \in \mathrm{BT}(s)$. By definition, $s \to^* s_1$ such that $N \leq_0 f(s_1)$. By confluence, there exists s'' such that $s' \to^* s''$ and $s_1 \to^* s''$, so we conclude by Lemma 20. ◀

Additionally, when $\mathcal{S}$ is confluent, $\mathrm{BT}(s)$ may be seen as a possibly infinite cut-free process, in the sense of taking the construct $!x(a).P$ coinductively:

▶ **Lemma 23.** *Let $\mathcal{S}$ be confluent. Then, $\mathrm{BT}(s)$ is either empty or an ideal w.r.t. $\leq_0$.*

Proof. Downward closure is immediate from the definition and transitivity of $\leq_0$. It remains to prove that $N_1, N_2 \in \mathrm{BT}(s)$ implies that there is $N \in \mathrm{BT}(s)$ such that $N_1, N_2 \leq_0 N$. We start by proving that, given an arbitrary process P, the poset $\{Q \mid Q \leq_0 P\}$ ordered by $\leq_0$ has binary suprema. Let τ_P be the rooted forest whose nodes are the boxes of P and such that there is an edge from R to S if S is a subprocess of R. By definition, $Q \leq_0 P$ iff Q is obtained by replacing some boxes of P with $\mathbf{0}$, so the poset $\{Q \mid Q \leq_0 P\}$ is isomorphic to the poset $\{\tau \mid \tau \text{ is a rooted subforest of } \tau_P\}$ ordered by rooted inclusion of forests, and the latter obviously has binary suprema.

Now, by definition, $N_1, N_2 \in \mathrm{BT}(s)$ means that $s \to^* s_1$ and $s \to^* s_2$ such that $N_1 \leq_0 f(s_1)$ and $N_2 \leq_0 f(s_2)$. By confluence, we have s' such that $s_1 \to^* s'$ and $s_2 \to^* s'$. By Lemma 20 and the fact that f is a morphism, $N_1, N_2 \leq_0 f(s')$, so we take N to be the supremum of N_1 and N_2. ◀

▶ **Definition 24** (Taylor expansion)**.** *Let s be an object of $\mathcal{S}$. A* Taylor approximation *of s is a Taylor process p such that $p \sqsubset f(s)$. The* Taylor expansion *of s, denoted by $\mathcal{T}(s)$, is the set of all Taylor approximations of s.*

The Taylor expansion of the Böhm tree *of s is the following set of Taylor processes:*

$$\mathcal{T}(\mathrm{BT}(s)) := \{n \sqsubset N \mid N \in \mathrm{BT}(s)\}.$$

Notice that, since Böhm approximations are cut-free, $\mathcal{T}(\mathrm{BT}(s))$ is actually a set of cut-free linear processes.

$$\frac{}{\vdash ; ; A^\perp, A} \qquad \frac{\vdash \Theta_1; \Theta_1'; \Gamma, A^\perp \quad \vdash \Theta_2; \Theta_2'; \Delta, A}{\vdash \Theta_1, \Theta_2; \Theta_1', \Theta_2'; \Gamma, \Delta} \qquad \frac{\vdash \Theta_1, A^\perp, \dots, A^\perp; \Theta_1'; \Gamma \quad \vdash \Theta_2; \Theta_2', A; \Delta}{\vdash \Theta_1, \Theta_2; \Theta_1', \Theta_2'; \Gamma, \Delta}$$

$$\frac{\vdash \Theta_1; \Theta_1'; \Gamma_1, A_1 \quad \dots \quad \vdash \Theta_n; \Theta_n'; \Gamma_n, A_n}{\vdash \Theta_1, \dots, \Theta_n; \Theta_1', \dots, \Theta_n'; \Gamma_1, \dots, \Gamma_n, A_1 \otimes \dots \otimes A_n} \qquad \frac{\vdash \Theta; \Theta'; \Gamma, A_1, \dots, A_n}{\vdash \Theta; \Theta'; \Gamma, A_1 \parr \dots \parr A_n}$$

$$\frac{\vdash \Theta; \Theta'; \Gamma, A}{\vdash \Theta, A; \Theta'; \Gamma} \qquad \frac{\vdash \Theta, A, \dots, A; \Theta'; \Gamma}{\vdash \Theta; \Theta'; \Gamma, ?A} \qquad \frac{\vdash \Theta; ; A}{\vdash \Theta; A;} \qquad \frac{\vdash \Theta; \Theta', A; \Gamma}{\vdash \Theta; \Theta'; \Gamma, !A}$$

Figure 2 Linear logic. Exchange rules (applicable within every segment of sequents) are implicit.

Given an arbitrary set of Taylor processes X, we define $\mathrm{NF}(X)$ as the set of cut-free forms of processes in X: $\mathrm{NF}(X) := \{n \text{ cut-free} \mid \exists p \in X \text{ such that } p \to^* n\}$.

▶ **Theorem 25** (Böhm-Taylor commutation, pulled back)**.** *For every object s of $\mathcal{S}$,*

$$\mathrm{NF}(\mathcal{T}(s)) = \mathcal{T}(\mathrm{BT}(s)).$$

Proof. Unfolding the definitions, we have $n \in \mathrm{NF}(\mathcal{T}(s))$ iff $n <\sqsubset f(s)$. Similarly, $n \in \mathcal{T}(\mathrm{BT}(s))$ iff $n \sqsubset< f(s)$: the forward implication is immediate from the definitions and the fact that f is a morphism; the converse follows from Lemma 21. We then conclude by Theorem 18. ◀

5 Linear Logic

5.1 Proofs as Processes

The sequent calculus of classical linear logic is presented in Fig. 2. We leave out additive connectives as they are unessential for our purposes. Sequents are divided in three segments, conveniently matching our process calculus. It is an exercise to show that this is equivalent to a more standard presentation, such as Girard's original one [27], in the following sense:

▶ **Proposition 26.** *A sequent $\vdash \Theta; \Theta'; \Gamma$ is provable in the sequent calculus of Fig. 2 iff the sequent $\vdash ?\Theta, !\Theta', \Gamma$ is provable in the sequent calculus of classical linear logic given in [27].*

Proof. Observe that the translation of each rule of Fig. 2 is derivable in Girard's calculus. Conversely, every rule of Girard's calculus is derivable in ours. Deriving contraction and promotion introduces cuts, using the derivability of $\vdash A; ; !A^\perp$. ◀

The sequent calculus of Fig. 2 may be decorated with processes and converted to a type system. This would be a "Curry-style" presentation of the correspondence between our processes and classical linear logic. We opt instead for a "Church-style" presentation:

▶ **Definition 27** (typed process)**.** *A* typed process *is a process in which every occurrence of name (except in binders) is decorated by a formula of linear logic, in such a way that:*

- *in $a^A \leftrightarrow b^B$, $B = A^\perp$;*
- *two occurrences of the same linear name are decorated by the same formula if one is subject and the other object, or by dual formulas if they are both subject or both object;*
- *in $\overline{a}^A\langle b_1^{B_1}, \dots, b_n^{B_n}\rangle$ (resp. $a^A\langle b_1^{B_1}, \dots, b_n^{B_n}\rangle$), $A = B_1 \otimes \dots \otimes B_n$ (resp. $A = B_1 \parr \dots \parr B_n$);*

$$\frac{a \neq b}{a \leftrightarrow b \rhd \Xi; ; a, b} \qquad \frac{P \rhd \Xi; \Xi_1'; \Gamma, a \quad Q \rhd \Xi; \Xi_2'; \Delta, a}{P \mid Q \rhd \Xi; \Xi_1', \Xi_2'; \Gamma, \Delta}$$

$$\frac{}{\overline{a}\langle \widetilde{b} \rangle \rhd \Xi; ; a, \widetilde{b}} \qquad \frac{P \rhd \Xi; \Xi'; \Gamma, \widetilde{b}}{a\langle \widetilde{b} \rangle \mid P \rhd \Xi; \Xi'; \Gamma, a} \qquad \frac{P \rhd \Xi, x; \Xi_1'; \Gamma \quad Q \rhd \Xi; \Xi_2', x; \Delta}{\nu x(P \mid Q) \rhd \Xi; \Xi_1', \Xi_2'; \Gamma, \Delta}$$

$$\frac{}{\overline{x}\langle a \rangle \rhd \Xi, x; ; a} \qquad \frac{P \rhd \Xi, x; \Xi'; \Gamma}{\overline{a}(x)P \rhd \Xi; \Xi'; \Gamma, a} \qquad \frac{P \rhd \Xi; ; a}{!x(a).P \rhd \Xi; x; } \qquad \frac{P \rhd \Xi; \Xi', x; \Gamma}{a(x)P \rhd \Xi; \Xi'; \Gamma, a}$$

Figure 3 Correct processes.

- *all occurrences of same polarity of a cartesian name are decorated by the same formula;*
- *in $\nu x P$, positive and negative occurrences of x (if any) are decorated by dual formulas;*
- *in $\overline{a}^A(x)P$ (resp. $a^A(x)P$), $A = ?B$ (resp. $A = !B$), where B is the formula decorating x in P, or is arbitrary if $x \notin \mathsf{fn}(P)$;*
- *in $\overline{x}^A\langle a^B \rangle$ and $x^A\langle a^B \rangle$, we have $A = B$; similarly, in $!x^A(a).P$, the decoration of a in P (which must appear) is A.*

The type *of a free occurrence of name as subject (resp. as object) is its decoration (resp. the negation of its decoration). We say that $\vdash \Theta; \Theta'; \Gamma$ is a* sequent associated with *P if Θ (resp. Θ', Γ) contains all types of all positive cartesian (resp. negative cartesian, linear) free occurrences of variables. This is unique up to a permutation of the occurrences, so in the sequel we will simply speak of "the" sequent associated with P.*

It is important to observe that typed does not imply logically correct: for example, $a^A \leftrightarrow a^{A^\perp}$ is typed, but its associated sequent is empty. In the literature on linear logic, there are *correctness criteria* [27, 13] for isolating "proof-like" objects. Here, we rely on an inductive presentation, given in Fig. 3, where a judgment $P \rhd \Xi; \Xi'; \Gamma$ means that P is correct and its free positive cartesian (resp. negative cartesian, linear) names are in Ξ (resp. Ξ', Γ). Observe that, in accordance with the yoga of linear logic, typability and correctness are independent notions: a correct process need not be typable.

▶ **Proposition 28.** *A sequent is provable in the sequent calculus of Fig. 2 iff it is associated with a correct typed process. Moreover, cut-free proofs correspond to cut-free processes.*

Proof. Both directions are by induction, on the last rule of the sequent calculus proof or on the last rule of the correctness derivation. ◀

So correct typed processes may be regarded as linear logic proofs, and reduction as a cut-elimination procedure. If one wished, confluence and termination could be proved for correct typed processes, but this is beyond the scope of this paper. We do observe, however, that Fig. 3 may easily be extended to include correctness for non-deterministic processes. It suffices to replace the last rule of Fig. 3 with the rule below on the left, and to add the rule below on the right:

$$\frac{P_1 \rhd \Xi; \Xi_1', x; \Gamma_1 \quad \dots \quad P_n \rhd \Xi; \Xi_n', x; \Gamma_n}{a(x)(P_1 \mid \cdots \mid P_n) \rhd \Xi; \Xi_1', \dots, \Xi_n'; \Gamma_1, \dots, \Gamma_n, a} \qquad \frac{}{x\langle a \rangle \rhd \Xi; x; a}$$

Correct (for this extended notion of correctness) typed processes are in correspondence with proofs of differential linear logic (with promotion but without zero and sums).

$$\frac{}{\mathbf{0} \vdash ; ;} \qquad \frac{P \vdash \Xi_1; \Xi_1'; \Gamma, [a : A] \quad Q \vdash \Xi_2; \Xi_2'; \Delta, [a : A^\perp]}{P \mid Q \vdash \Xi_1, \Xi_2; \Xi_1', \Xi_2'; \Gamma, \Delta} \qquad \frac{}{a \leftrightarrow b \vdash ; ; a : A, b : A^\perp}$$

$$\frac{}{\overline{a}\langle b_1, \ldots, b_n\rangle \vdash ; ; a : \bigotimes_i A_i, b_1 : A_1^\perp \ldots, b_n : A_n^\perp}$$

$$\frac{}{a\langle b_1, \ldots, b_n\rangle \vdash ; ; a : \parr_i A_i, b_1 : A_1^\perp \ldots, b_n : A_n^\perp}$$

$$\frac{P \vdash \Xi_1, x : A_1, \ldots, x : A_n; \Xi_1'; \Gamma \quad Q \vdash \Xi_2; \Xi_2', x : A_1^\perp, \ldots, x : A_n^\perp; \Delta}{\nu x(P \mid Q) \vdash \Xi_1, \Xi_2; \Xi_1', \Xi_2'; \Gamma, \Delta}$$

$$\frac{P \vdash \Xi, x : A_1, \ldots, x : A_n; \Xi'; \Gamma}{\overline{a}(x)P \vdash \Xi; \Xi'; \Gamma, a : \bigvee_i A_i}$$

$$\frac{P_1 \vdash \Xi_1; \Xi_1', x : A_1; \Gamma_1 \quad \ldots \quad P_n \vdash \Xi_n; \Xi_n', x : A_n; \Gamma_n}{a(x)(P_1 \mid \cdots \mid P_n) \vdash \Xi_1, \ldots, \Xi_n; \Xi_1', \ldots, \Xi_n'; \Gamma_1, \ldots, \Gamma_n, a : \bigwedge_i A_i}$$

$$\frac{}{\overline{x}\langle a\rangle \vdash x : A; ; a : A^\perp} \qquad \frac{}{x\langle a\rangle \vdash ; x : A; a : A^\perp}$$

$$\frac{P \vdash \Xi_1; ; a : A_1 \quad \ldots \quad P \vdash \Xi_n; ; a : A_n}{!x(a).P \vdash \Xi_1, \ldots, \Xi_n; x : A_1, \ldots, x : A_n;}$$

Figure 4 Intersection types. In the second rule at the top, the declarations $a : A$ and $a : A^\perp$ are either both present or both absent (the rule is a cut in the first case, a mix in the second).

One may also add correctness of the so-called *mix rules*:

$$\frac{}{\mathbf{0} \rhd ; ;} \qquad \frac{P \rhd \Xi; \Xi_1'; \Gamma \quad Q \rhd \Xi; \Xi_2'; \Delta}{P \mid Q \rhd \Xi; \Xi_1', \Xi_2'; \Gamma, \Delta}$$

This more general notion of correctness, for non-deterministic processes and with mix rules, is the one we will consider in the next section.

5.2 Intersection Types

Intersection types are defined as follows:

$$A, B ::= X \mid X^\perp \mid A_1 \otimes \cdots \otimes A_n \mid A_1 \parr \cdots \parr A_n \mid A_1 \wedge \cdots \wedge A_n \mid A_1 \vee \cdots \vee A_n.$$

Duality (*i.e*, linear negation) is defined as usual, with $\wedge$ dual to $\vee$.

An *intersection type judgment* is of the form $P \vdash \Xi; \Xi'; \Gamma$ where Ξ and Ξ' (resp. Γ) contain type declarations of the form $x : A$ (resp. $a : A$) with x a cartesian name (resp. a a linear name) and A an intersection type. The same cartesian name may appear in multiple type declarations in Ξ and Ξ', even multiple times with the same type (this means that we are considering non-idempotent intersection types). By contrast, a linear name may only be declared once in Γ. The intersection type system for processes is given in Fig. 4.

A *typed Taylor process* is defined as in Definition 27, with the addition of the constraints that in $\overline{a}^A\langle\!\langle b_1^{B_1}, \dots, b_n^{B_n}\rangle\!\rangle$ (resp. $a^A\langle\!\langle b_1^{B_1}, \dots, b_n^{B_n}\rangle\!\rangle$) we have $A = B_1 \vee \cdots \vee B_n$ (resp. $A = B_1 \wedge \cdots \wedge B_n$). Taylor processes do not have cartesian names, so the associated sequent of a typed Taylor process is of the form $\vdash;;\Gamma$, which we just write $\vdash \Gamma$. If p is a typed Taylor process, we write p^- for the underlying process, without the decorations. In the following, correctness is meant in the generalized sense of the end of §5.1 (with mix).

▶ **Lemma 29.** *If p is a correct typed Taylor process, then p^- has a cut-free form.*

Proof. Taylor processes are linear, so reduction always terminates (Lemma 11). Therefore, it is enough to show that there are no irreducible cuts. This is straightforward: irreducible cuts are incorrect or not typable and reduction preserves correctness and type decorations. ◀

The following is a reformulation of the results of [32], where a general connection between Taylor approximations and intersection types is described in detail. We state it without proof, but one may intuitively justify the result by observing that Fig. 4 is a superposition of Fig. 1 and Fig. 3 (with the additional rules given at the end of §5.1), annotated with types.

▶ **Proposition 30.** *The judgment $P \vdash \Xi; \Xi'; \Gamma$ is derivable in the system of Fig. 4 iff there exists a correct typed Taylor process p whose associated sequent is $\vdash \Xi, \Xi', \Gamma$ such that $p^- \sqsubset P$.*

▶ **Lemma 31** (subject expansion)**.** *If $Q \vdash \Xi; \Xi'; \Gamma$ is derivable and P is correct such that $P \to Q$, then $P \vdash \Xi; \Xi'; \Gamma$ is derivable.*

Proof. Modulo Proposition 30, this is Lemma 13, except that we need to add type decorations. Correctness of P ensures that the pullback approximation is also correct. ◀

We say that a context is *shallow* if the hole does not appear inside a box. *Shallow reduction* is defined by modifying Definition 7 as follows: the reduction rule $\mathsf{cod}_!$ is discarded, c is restricted to the case $\mathsf{C} = Q \mid \{-\}$, with Q arbitrary, and the reduction rules are only closed under shallow contexts. We write $\to_0$ for shallow reduction. A process is said to be *shallow cut-free* if all of its cuts, if any, are inside boxes.

▶ **Lemma 32** (progress)**.** *Let $P \vdash \Xi; \Xi'; \Gamma$ be derivable and let p be the associated typed Taylor process according to Proposition 30. Then, either P is shallow cut-free or $P \to_0^* Q$ and there is a derivation $Q \vdash \Xi; \Xi'; \Gamma$ whose associated Taylor process q is such that $p^- \to q^-$.*

Proof. By inspection of Fig. 1, we see that a shallow cut in P yields a cut in p. As observed in Lemma 29, such a cut cannot be irreducible, so we have $p \to q$ by reducing it. Removing type annotations, we have $p^- \sqsubset P$ and $p^- \to q^-$, so we apply Lemma 15 and Proposition 30. ◀

Observe that Lemma 32 does not hold for general cuts: since boxes may be approximated by $\mathbf{0}$, P may contain a cut inside a box which is invisible to the intersection type derivation.

▶ **Theorem 33.** *A process P is typable as in Fig. 4 iff it is correct (in the generalized sense of the end of §5.1) and $P \to_0^* P_0$ with P_0 shallow cut-free.*

Proof. Let P be typable. Correctness is immediate from the typing rules: they are essentially a decoration of Fig. 3 plus the extra correctness rules at the end of §5.1. We need to show that P reduces to a shallow cut-free process. Let p be the typed Taylor approximation given by Proposition 30. We reason by induction on the size of p^-, as defined in the proof of Lemma 11. We apply Lemma 32 and either conclude immediately because P is shallow cut-free or obtain $P \to^* Q$ with Q typable with an associated approximation q such that $p^- \to q^-$. This implies that the size of q^- is strictly smaller than the size of p^-, so we conclude by the induction hypothesis.

Suppose now that $P \to_0^* P_0$ with P_0 shallow cut-free. It is straighforward to prove, by induction on P_0, that P_0 is typable (intuitively, we approximate all boxes by 0 and then typability is guaranteed by correctness and absence of cuts), so we conclude by Lemma 31. ◀

If $f : \mathcal{S} \to \mathcal{P}roc$ is a morphism of reduction systems, then we may say that an object of s is *typable in intersection types* if $f(s)$ is typable according to Fig. 4.

Let $\mathcal{P}roc_0$ be the reduction system with processes as objects but with $\to_0^*$ as preorder. In the following, we consider a reduction system $\mathcal{S}$ with a distinguished set of objects suggestively called "normal". We say that an embedding $f : \mathcal{S} \to \mathcal{P}roc_0$ is *sound* if

- for every object s, $f(s)$ is correct (in the generalized sense);
- for every object s_0, s_0 is normal iff $f(s_0)$ is shallow cut-free.

▶ **Theorem 34.** *Let $f : \mathcal{S} \to \mathcal{P}roc_0$ be a sound embedding. Then, an object s of $\mathcal{S}$ is typable in intersection types iff $s \to^* s_0$ with s_0 normal.*

Proof. Suppose that s is typable, which means that $f(s)$ is. By Theorem 33, $f(s) \to_0^* P_0$ with P_0 shallow cut-free. Since f is an embedding, we have $P_0 \to_0^* f(s_0)$ such that $s \to^* s_0$. But P_0 is shallow cut-free, so $f(s_0) = P_0$ and we conclude s_0 normal by soundness.

Suppose now that $s \to^* s_0$ with s_0 normal. This implies $f(s) \to_0^* f(s_0)$. By soundness, $f(s_0)$ is correct and shallow cut-free, so it is typable. By soundness and Lemma 31, $f(s)$ is typable, so s is typable by definition. ◀

6 Applications

6.1 Call by Push Value

It is known that Paul Levy's call-by-push-value [30] may be expressed in intuitionistic linear logic [19], yielding the *bang-calculus* [21]. We use a recent reformulation due to Bucciarelli et al. [8], which allows us to show at the same time how Accattoli and Kesner's explicit substitutions "at a distance" may be handled painlessly. Terms are defined by

$$t, u ::= x \mid \lambda x.t \mid tu \mid !t \mid \operatorname{der} t \mid t[x := u],$$

where x ranges over a countable set of variables, which, for convenience, we take to be the set of cartesian names. Contexts are defined as expected (add a hole $\{-\}$ to the above definition). We use the notation $t[-]$ for a term of the form $t[x_1 := u_1] \cdots [x_n := u_n]$ (n may be zero). The constructor $!(-)$ binds more strongly than binary constructors, *i.e*, $!tu$ and $!t[x := u]$ are to be understood as $(!t)u$ and $(!t)[x := u]$, respectively. Reduction is the contextual closure of the following rules:

$$(\lambda x.t)[-]u \;\to\; t[x := u][-] \qquad t[x := !u[-]] \;\to\; t\{u/x\}[-] \qquad \operatorname{der}(!t[-]) \;\to\; t[-]$$

where $t\{u/x\}$ denotes, as usual, the capture-free substitution of u to all free occurrences of x in t. This induces a reduction system $\Lambda_!$. We inductively define a family of maps $(\!|-|\!)a : \Lambda_! \to \mathcal{P}roc$ parametric in a linear name a:

$$\begin{aligned}
(\!|x|\!)a &:= \overline{x}\langle a\rangle & (\!|\lambda x.t|\!)a &:= a\langle c, d\rangle \mid \overline{c}(x)(\!|t|\!)d \\
(\!|tu|\!)a &:= (\!|t|\!)b \mid \overline{b}\langle c, a\rangle \mid (\!|u|\!)c & (\!|!t|\!)a &:= a(z)!z(b).(\!|t|\!)b \\
(\!|\operatorname{der} t|\!)a &:= \overline{c}(z)\overline{z}\langle a\rangle \mid (\!|t|\!)c & (\!|t[x := u]|\!)a &:= \overline{b}(x)(\!|t|\!)a \mid (\!|u|\!)b.
\end{aligned}$$

The paper [8] also introduces *weak reduction* for the bang-calculus, which does not reduce under $!(-)$. We denote by $\Lambda_!^w$ the corresponding reduction system. In this system, we take normal forms to be terms whose redexes and clashes (undesirable configurations defined in [8]) only appear under a $!(-)$.

▶ **Proposition 35.** *For any linear name a, $(\!|-|\!)a$ is an embedding. Moreover, considered as a map $\Lambda_!^w \to \mathcal{P}roc_0$, it is a sound embedding.*

With a bit of work (which we spare the reader), the results of §4.3 may be formulated directly in the syntax of the bang-calculus. Böhm trees are as expected: given a bang-calculus term t, if weak reduction for t does not terminate, then $\mathrm{BT}(t) = \bot$. Otherwise, it terminates on a term of the form $\mathsf{C}\{!u_1, \dots, !u_n\}$ where C is a multi-hole context containing no $!(-)$. Then, $\mathrm{BT}(t) = \mathsf{C}\{!\mathrm{BT}(u_1), \dots, !\mathrm{BT}(u_n)\}$. Notice that $\bot$ and $!\bot$ are different Böhm trees.

The Taylor expansion of call by push value has already been defined and studied in [10]. Here, we find a reformulation in the context of the bang-calculus with explicit substitutions. Taylor approximation terms are defined as follows:

$$r, s ::= a \mid \lambda\langle\widetilde{a}\rangle.r \mid rs \mid \langle r_1, \dots, r_n\rangle \mid \mathrm{der}\, r \mid r[\langle\widetilde{a}\rangle := s],$$

where a ranges over linear variables, *i.e*, no variable occurs twice and every variable of $\widetilde{a}$ in binders $\lambda\langle\widetilde{a}\rangle.t$ or $t[\langle\widetilde{a}\rangle := u]$ must occur free in t. The reduction rules are as follows:

$$(\lambda\langle\widetilde{a}\rangle.r)s \to r[\langle\widetilde{a}\rangle := s] \qquad r[\langle\widetilde{a}\rangle := \langle\widetilde{s}\rangle] \to r\{\widetilde{s}/\widetilde{a}\} \qquad \mathrm{der}\langle r\rangle \to r,$$

with the condition that, in the second rule, $|\widetilde{a}| = |\widetilde{s}|$. The approximation relation is defined using judgments $\Xi \vdash r \sqsubset t$ with Ξ consisting of declarations of the form $a \sqsubset x$, with no linear variable appearing twice in Ξ. The relation is defined as follows:

$$\frac{}{a \sqsubset x \vdash a \sqsubset x} \qquad \frac{\Xi, \widetilde{a} \sqsubset x \vdash r \sqsubset t}{\Xi \vdash \lambda\langle\widetilde{a}\rangle.r \sqsubset \lambda x.t} \qquad \frac{\Xi \vdash r \sqsubset t \quad \Upsilon \vdash s \sqsubset u}{\Xi, \Upsilon \vdash rs \sqsubset tu}$$

$$\frac{\Xi_1 \vdash r_1 \sqsubset t \quad \dots \quad \Xi_n \vdash r_n \sqsubset t}{\Xi_1, \dots, \Xi_n \vdash \langle\widetilde{r}\rangle \sqsubset\, !t} \qquad \frac{\Xi \vdash r \sqsubset t}{\Xi \vdash \mathrm{der}\, r \sqsubset \mathrm{der}\, t} \qquad \frac{\Xi, \widetilde{a} \sqsubset x \vdash r \sqsubset t \quad \Upsilon \vdash s \sqsubset u}{\Xi, \Upsilon \vdash r[\langle\widetilde{a}\rangle := s] \sqsubset t[x := u]}$$

By Proposition 35 and Theorem 25, we know that the above Böhm trees and Taylor approximations interact well.

Proposition 35 also entails the results of §5.2. We will not detail them here, but these allow us to obtain for free the intersection type system of [8], along with the property that it characterizes terms with weakly clash-free normal forms.

Both [10] and [8] consider the well-known embeddings of call-by-name and call-by-value λ-calculus in call by push value, and extrapolate from these embeddings suitable notions of Taylor approximations and intersection type systems for call-by-name and call-by-value, recovering the results of [23, 25, 18, 29] and [26, 14, 15, 18]. In our setting, these are embeddings in the technical sense of Definition 1, and since embeddings compose, we also recover these results in a uniform manner. Additionally, we recover usual Böhm trees [4] and the call-by-value Böhm trees of [29], as well as the related commutation theorems [23, 25, 29].

6.2 Classical Logic

The *stack calculus* [9] is a simple calculus for classical computation, embedding the $\lambda\mu$-calculus. Its syntax has *stacks* π, *terms* t and *processes* P, defined as follows:

$$\pi ::= \alpha \mid t \cdot \pi \mid \mathsf{tl}(\pi) \qquad t ::= \mu\alpha.P \mid \mathsf{hd}(\pi) \qquad P ::= \langle t, \pi\rangle,$$

where α ranges over a countably infinite set of *stack variables*, which, for technical convenience, we take to be a subset of cartesian names of our processes. The construct $\mu\alpha$ is a binder. Reduction is defined as follows:

$$\langle\mu\alpha.P, \pi\rangle \to P\{\pi/\alpha\} \qquad \mathsf{hd}(t \cdot \pi) \to t \qquad \mathsf{tl}(t \cdot \pi) \to \pi.$$

Let $\mathcal{S}tk$, $\mathcal{T}erm$ amd $\mathcal{S}tkProc$ be the reduction systems induced by the above definitions, with stacks, terms and processes as objects, respectively. We define two families of maps $(\!|-|\!)a : \mathcal{S}tk \to \mathcal{P}roc$ and $(\!|-|\!)a : \mathcal{T}erm \to \mathcal{P}roc$ parametric in a linear name a, as well as a map $(\!|-|\!) : \mathcal{S}tkProc \to \mathcal{P}roc$, as follows:

$$\begin{array}{rlrl}
(\!|\alpha|\!)a &:= \overline{\alpha}\langle a\rangle & (\!|t\cdot\pi|\!)a &:= \overline{a}\langle b,c\rangle \mid b(x)!x(d).(\!|t|\!)d \mid c(y)!y(e).(\!|\pi|\!)e \\
(\!|\mu\alpha.P|\!)a &:= \overline{a}(\alpha)(\!|P|\!) & (\!|\mathsf{hd}(\pi)|\!)a &:= (\!|\pi|\!)b \mid b\langle c,d\rangle \mid \overline{c}(x)\overline{d}(y)(\overline{x}\langle a\rangle) \\
(\!|\langle t,\pi\rangle|\!) &:= (\!|t|\!)a \mid a(x)!x(b).(\!|\pi|\!)b & (\!|\mathsf{tl}(\pi)|\!)a &:= (\!|\pi|\!)b \mid b\langle c,d\rangle \mid \overline{c}(x)\overline{d}(y)(\overline{y}\langle a\rangle).
\end{array}$$

▶ **Proposition 36.** *The above maps are embeddings.*

As mentioned in the introduction, the encoding of the $\lambda\mu$-calculus in the stack calculus is not an embedding in our technical sense, so we cannot directly apply our results to the $\lambda\mu$-calculus. Nevertheless, now we do have a working theory of Böhm trees and Taylor expansion for a calculus Curry-Howard-isomorphic to classical logic, which is a novelty as far as we know (in the context of the $\lambda\mu$-calculus, it is mentioned as an open question in [2]). We leave the investigation of this theory, in particular the significance of Böhm trees, to future work.

6.3 Concurrent Computation

The asynchronous polyadic π-calculus is defined as follows:

$$P,Q ::= \mathbf{0} \;\Big|\; P \mid Q \;\Big|\; \nu x P \;\Big|\; \overline{x}\langle\widetilde{y}\rangle \;\Big|\; !x(\widetilde{y}).P,$$

where we suppose names to be cartesian names of our processes. Structural congruence and reduction are standard, with the rules

$$\overline{x}\langle\widetilde{y}\rangle \mid !x(\widetilde{z}).P \to P\{\widetilde{y}/\widetilde{z}\} \mid !x(\widetilde{z}).P \qquad\qquad \nu x(!x(\widetilde{y}).P \mid Q) \to \nu x Q$$

with the proviso, in the second rule, that x does not occur as subject of an output in Q.

We consider here the *hyperlocalized* variant of the calculus [12], which is defined by restricting to processes such that, in $!x(\widetilde{y}).P$, no free name of P occurs as subject of an input. Also, reduction is allowed only under a restriction. As shown in [12], this is a reasonably expressive calculus, with full non-determinism, locks, etc.

Let Π be the reduction system corresponding to the above calculus. Using the notation $(z \Rightarrow y) := !z(c).\overline{y}\langle c\rangle$ (as a process of $\mathcal{P}roc$), we define a map $(\!|-|\!) : \Pi \to \mathcal{P}roc_0$ by letting

$$\begin{array}{rl}
(\!|\overline{x}\langle y_1,\dots,y_n\rangle|\!) &:= \overline{x}\langle a\rangle \mid \overline{a}\langle b_1,\dots,b_n\rangle \mid b_1(z_1)(z_1\Rightarrow y_1) \mid \cdots \mid b_n(z_n)(z_n\Rightarrow y_n) \\
(\!|!x(y_1,\dots,y_n).P|\!) &:= !x(a).(a\langle b_1,\dots,b_n\rangle \mid \overline{b}_1(y_1)\cdots\overline{b}_n(y_n)(\!|P|\!))
\end{array}$$

and by making $(\!|-|\!)$ act homomorphically on $\mathbf{0}$, parallel composition and restriction.

▶ **Proposition 37.** *The map $(\!|-|\!)$ is an embedding.*

Unfortunately, the above embedding is not sound with respect to any reasonable notion of normal form for the π-calculus, because $(\!|P|\!)$ is not necessarily correct (processes may have all sorts of vicious cycles). However, this does not prevent from taking the intersection type system of §5.2 and use it as a starting point for coming up with a working intersection type system for the hyperlocalized π-calculus. This is exactly the genesis of the paper [12].

References

1 Beniamino Accattoli. Exponentials as substitutions and the cost of cut elimination in linear logic. *Log. Methods Comput. Sci.*, 19(4), 2023.

2 Davide Barbarossa. Resource approximation for the $\lambda\mu$-calculus. In *Proceedings of LICS*, pages 27:1–27:12, 2022.

3 Davide Barbarossa and Giulio Manzonetto. Taylor subsumes Scott, Berry, Kahn and Plotkin. *Proc. ACM Program. Lang.*, 4(POPL):1:1–1:23, 2020.

4 Henk Barendregt. The type free lambda calculus. In Jon Barwise, editor, *Handbook of Mathematical Logic*, volume 90 of *Studies in Logic and the Foundations of Mathematics*, pages 1091–1132. Elsevier, 1977.

5 Henk Barendregt. *The lambda calculus - its syntax and semantics*, volume 103 of *Studies in logic and the foundations of mathematics*. North-Holland, 1985.

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

7 Gérard Boudol. The lambda-calculus with multiplicities (abstract). In *Proceedings of CONCUR*, volume 715 of *Lecture Notes in Computer Science*, pages 1–6. Springer, 1993.

8 Antonio Bucciarelli, Delia Kesner, Alejandro Ríos, and Andrés Viso. The bang calculus revisited. *Inf. Comput.*, 293:105047, 2023.

9 Alberto Carraro, Thomas Ehrhard, and Antonino Salibra. The stack calculus. In *Proceedings of LSFA*, volume 113 of *EPTCS*, pages 93–108, 2012.

10 Jules Chouquet and Christine Tasson. Taylor expansion for call-by-push-value. In *Proceedings of CSL*, volume 152 of *LIPIcs*, pages 16:1–16:16, 2020.

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

12 Ugo Dal Lago, Marc de Visme, Damiano Mazza, and Akira Yoshimizu. Intersection types and runtime errors in the pi-calculus. *Proceedings of the ACM on Programming Languages*, 3(POPL:7), 2019.

13 Vincent Danos and Laurent Regnier. The structure of multiplicatives. *Arch. Math. Log.*, 28(3):181–203, 1989.

14 Daniel de Carvalho. *Sémantiques de la logique linéaire et temps de calcul*. Ph.D. thesis, Université de la Méditerranée–Aix-Marseille 2, 2007.

15 Daniel de Carvalho. Execution time of λ-terms via denotational semantics and intersection types. *Math. Struct. Comput. Sci.*, 28(7):1169–1203, 2018.

16 Thomas Ehrhard. On Köthe sequence spaces and linear logic. *Math. Struct. Comput. Sci.*, 12(5):579–623, 2002.

17 Thomas Ehrhard. Finiteness spaces. *Math. Struct. Comput. Sci.*, 15(4):615–646, 2005.

18 Thomas Ehrhard. Collapsing non-idempotent intersection types. In *Proceedings of CSL*, volume 16 of *LIPIcs*, pages 259–273, 2012.

19 Thomas Ehrhard. Call-by-push-value from a linear logic point of view. In *Proceedings of ESOP*, volume 9632 of *Lecture Notes in Computer Science*, pages 202–228, 2016.

20 Thomas Ehrhard. An introduction to differential linear logic: proof-nets, models and antiderivatives. *Math. Struct. Comput. Sci.*, 28(7):995–1060, 2018.

21 Thomas Ehrhard and Giulio Guerrieri. The bang calculus: an untyped lambda-calculus generalizing call-by-name and call-by-value. In *Proceedings of PPDP*, pages 174–187, 2016.

22 Thomas Ehrhard and Olivier Laurent. Acyclic solos and differential interaction nets. *Log. Methods Comput. Sci.*, 6(3), 2010.

23 Thomas Ehrhard and Laurent Regnier. Böhm trees, Krivine's machine and the Taylor expansion of lambda-terms. In *Proceedings of CiE 2006*, volume 3988 of *Lecture Notes in Computer Science*, pages 186–197, 2006.

24 Thomas Ehrhard and Laurent Regnier. Differential interaction nets. *Theor. Comput. Sci.*, 364(2):166–195, 2006.

25 Thomas Ehrhard and Laurent Regnier. Uniformity and the Taylor expansion of ordinary lambda-terms. *Theor. Comput. Sci.*, 403(2-3):347–372, 2008.

26 Philippa Gardner. Discovering needed reductions using type theory. In *Proceedings of TACS*, pages 555–574, 1994.

27 Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987.

28 Kohei Honda and Olivier Laurent. An exact correspondence between a typed pi-calculus and polarised proof-nets. *Theor. Comput. Sci.*, 411(22-24):2223–2238, 2010.

29 Axel Kerinec, Giulio Manzonetto, and Michele Pagani. Revisiting call-by-value Böhm trees in light of their Taylor expansion. *Log. Methods Comput. Sci.*, 16(3), 2020.

30 Paul Blain Levy. *Call-By-Push-Value: A Functional/Imperative Synthesis*, volume 2 of *Semantics Structures in Computation*. Springer, 2004.

31 Damiano Mazza. The true concurrency of differential interaction nets. *Math. Struct. Comput. Sci.*, 28(7):1097–1125, 2018.

32 Damiano Mazza, Luc Pellissier, and Pierre Vial. Polyadic approximations, fibrations and intersection types. *Proceedings of the ACM on Programming Languages*, 2(POPL:6), 2018.

33 Michele Pagani and Paolo Tranquilli. The conservation theorem for differential nets. *Math. Struct. Comput. Sci.*, 27(6):939–992, 2017.

34 Paolo Tranquilli. Confluence of pure differential nets with promotion. In *Proceedings of CSL*, volume 5771 of *Lecture Notes in Computer Science*, pages 500–514, 2009.

35 Takeshi Tsukada, Kazuyuki Asada, and C.-H. Luke Ong. Generalised species of rigid resource terms. In *Proceedings of LICS*, pages 1–12, 2017.

homotopy.io: A Proof Assistant for Finitely-Presented Globular n-Categories

Nathan Corbyn
University of Oxford, Oxford, UK

Lukas Heidemann
University of Oxford, Oxford, UK

Nick Hu
University of Oxford, Oxford, UK

Chiara Sarti
University of Cambridge, Cambridge, UK

Calin Tataru
University of Cambridge, Cambridge, UK

Jamie Vicary
University of Cambridge, Cambridge, UK

Abstract

We present the proof assistant homotopy.io for working with finitely-presented semistrict higher categories. The tool runs in the browser with a point-and-click interface, allowing direct manipulation of proof objects via a graphical representation. We describe the user interface and explain how the tool can be used in practice. We also describe the essential subsystems of the tool, including collapse, contraction, expansion, typechecking, and layout, as well as key implementation details including data structure encoding, memoisation, and rendering. These technical innovations have been essential for achieving good performance in a resource-constrained setting.

2012 ACM Subject Classification Software and its engineering → Software notations and tools

Keywords and phrases Higher category theory, proof assistant, string diagrams

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.30

Related Version *Full Version*: `https://arxiv.org/abs/2402.13179`

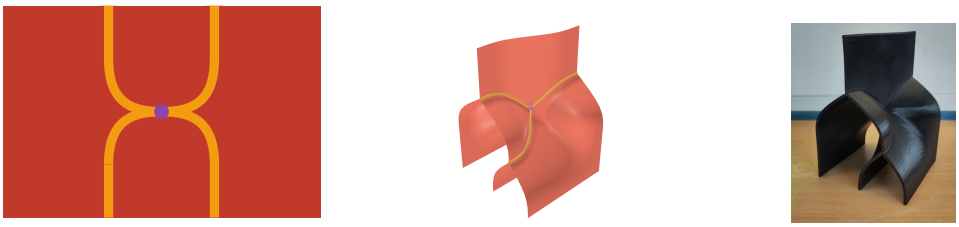
Supplementary Material *Software (Source Code)*: `https://github.com/homotopy-io/homotopy-rs` archived at `swh:1:dir:836bb913a5b2d2556f124f1a260a631a1a7f7387`
Audiovisual (Video Tutorial): `https://homotopy.io/braiding-example` [1]

Funding *Nathan Corbyn*: EPSRC Industrial CASE Studentship
Nick Hu: EPSRC Doctoral Training Partnership Scholarship [grant number 2218955]

Acknowledgements The authors would like to thank Anastasia Courtney, Yulong Huang, and Jasper Parish for their contributions during their summer internships, Akvilė Valentukonytė and Klaudia Urbanska for their contributions during their undergraduate projects, and Manuel Araújo, Wilf Offord, and Hao Xu for extensive testing of the tool and valuable feedback. We are also grateful to the students of the "Categorical Quantum Mechanics" course at Oxford, and the "Advanced Topics in Category Theory" course at Cambridge for testing and feedback.

1 Introduction

Higher category theory [26, 34] is a branch of mathematics that now has a wide range of applications, in areas as diverse as logic [18, 41], quantum field theory [3, 36], and geometry [27]. For working practically with higher categories, string diagrams are an



(a) 2D projection. **(b)** 3D projection. **(c)** 3D print.

Figure 1 The associator 3-diagram shown in both 2D and 3D (link to online workspace).

increasingly popular technique, introduced for monoidal categories by Joyal and Street [23], and since extended into higher dimensions by a range of authors [5, 7, 11, 13]. In a higher-categorical setting, string diagrams take the form of higher-dimensional manifold-like structures, which can efficiently encode complex compositional information. However, these structures can be hard to visualise, manipulate, or represent in research articles, limiting their effectiveness.

Our tool aims to bridge this gap, allowing string diagrams to become a practical technique for working higher category theorists. It runs in the browser, giving a low barrier-to-entry, and allows direct construction and manipulation of graphical representations of n-dimensional categorical structures, which we call *n-diagrams*, by direct point-and-click manipulation with the mouse or touch interface.

In this article, we give an overview of the user interface, and describe the following major subsystems, which give the tool its range of functionality.

- *Collapse* acts on a combinatorial n-diagram, constructing a quotient geometry by identifying points which can be considered topologically equivalent.
- *Contraction* allows a region of an n-diagram to be geometrically contracted to a point, yielding a homotopy that is itself encoded by an $(n+1)$-diagram; the collapse algorithm gives its base case. This is the major mechanism for constructing all nontrivial diagrams in the theory.
- *Expansion* provides a limited converse to contraction, defined on a diagram with at least two vertices at the same height, with the effect that one vertex is moved to an adjacent height. The resulting diagram will always contract to yield the original diagram.
- *Typechecking* analyses an instance of our n-diagram data structure, and decides whether it represents a valid composite n-morphism in a free higher category.
- *Layout* generates a set of linear constraints representing the necessary coordinate relationships between all the parts of a diagram (such as the vertices, wires, and regions), which can be passed to a linear solver, and used by the rendering pipeline.

We also examine two significant aspects of the implementation.

- *Memoisation* is necessary since the n-diagrams stored by the tool have an intricate recursive structure, which in principle encodes all sub-k-diagrams for $k < n$. If this data was stored separately in memory, the resource requirements of the proof assistant would grow exponentially with diagram dimension. Memoisation ensures each logically-distinct k-diagram is stored only once in memory.
- *Rendering* is a complex pipeline that produces suitable output on the screen (see Figure 1); we use SVG for output in dimension 0, 1, and 2, and WebGL for output in dimension 3 and 4. A subdivision algorithm is necessary to produce visually appealing output. We also render to STL for 3D printing, and to TikZ for convenient diagram export.

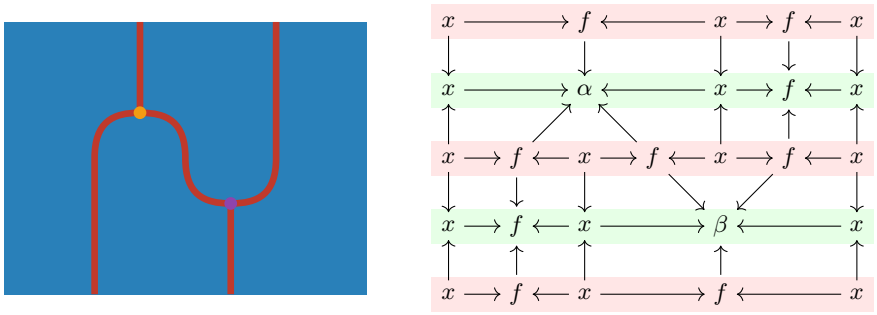


Figure 2 A string diagram and the corresponding zigzag encoding.

The version of the tool presented in this article is a pre-release version, available here:

`https://beta.homotopy.io`

It is written in the Rust programming language, and compiled to WebAssembly to run in the web browser. The implementation is available under a free open-source licence on GitHub.

1.1 Mathematical Context

Our proof assistant implements the theory of higher categories known as *associative n-categories*, due to Dorn, Douglas and Vicary [12, 33]. This model is *globular*, in the sense that for $n \geq 2$ the boundary data of any n-cell satisfies the globularity condition: the source of the source equals the source of the target, and the target of the source equals the target of the target. It is also strictly associative and unital, while retaining weak interchangers; in this sense it is a *semistrict* theory. It is conjectured that every weak higher category is equivalent to an associative n-category, although the proof of this remains out of reach.

The proof assistant allows users to build composite cells in the graphical language for freely generated semistrict globular n-categories which are free on a *signature*, a list of variables of specified types. For example, to define a monad-like structure, starting from the empty signature, we might first add an object x, followed by a 1-cell $f\colon x \to x$, followed in turn by a multiplication 2-cell $\mu\colon f \circ f \to f$, which we interpret as the monad multiplication.

The tool allows generators of non-zero dimension to be optionally tagged as invertible; this allows the user to work with directed higher categories (if no generators are tagged), higher groupoids (if all generators are tagged), or more general structures. The resulting invertible structure is coherent, meaning that it automatically satisfies the necessary higher-dimensional constraints, such as the adjunction equations.

In the implementation, n-cells are represented combinatorially as *n-diagrams*, simple inductive data structures which allow us to represent the mathematical zigzag construction [33]. We depict this in Figure 2, which illustrates the encoding of a 2-diagram (drawn on the left) in terms of an iterated sequence of cospans (drawn on the right). Here, a zigzag is a sequence of cospans taking value in some category of labels. We distinguish the *singular heights* drawn in green, where vertices might appear, from the *regular heights* drawn in red which are adjacent. Since zigzags and their morphisms themselves form a category, this construction can be iterated, and in this way higher-dimensional diagrams can be represented. This inductive nature of the construction is an essential requirement for our proof assistant: it allows us to describe n-cells as algebraic data types, as we will detail in Section 4, and to structure our key algorithms of Section 5 as recursive procedures.

Table 1 Comparison of existing tools for string diagrams.

tool	generality	interactive	invertibility	visualisation
homotopy.io	n-categories	✓	✓	up to 4D
Cartographer [37]	symmetric monoidal categories	✓	✗	2D
DisCoPy [10]	monoidal categories	✗	✗	2D
Globular [6]	4-categories	✓	partial	2D
rewalt [16]	n-categories	✗	partial	2D
sd-visualiser [20]	traced cartesian closed categories	✓	✗	2D
Quantomatic [24]	compact closed categories	✓	✗	2D
wiggle.py [9]	monoidal 2-categories	✗	✗	up to 3D

1.2 Related Work

There are many tools for higher categories that use string diagrams as a visualisation method, and we summarise several of them in Table 1. Each takes a different categorical perspective, and is focused on a particular formalism. The most closely related tool is Globular, the direct precursor to homotopy.io, which was limited to 4-categories and lacked full support for coherent inverses. One other tool, rewalt, also allows manipulations at the level of n-categories for arbitrary n. Both homotopy.io and rewalt implement semistrict n-categories, but they have different notions of semistrictness: homotopy.io has strict associators and unitors, but weak interchanges; whereas rewalt has strict interchangers and associators, with weak identities. This difference of approach means that the corresponding notions of string diagram are quite different in each tool. The homotopy.io tool is the first string diagram proof assistant that can handle coherent invertible generators in all dimensions, an aspect it shares with traditional type-theoretic approaches to higher category theory, such as homotopy type theory [41], interpreted via proof assistants such as Agda or Coq.

This paper is the first detailed description of homotopy.io, with previous works focused on aspects of the theoretical foundations [33, 17, 39, 35, 40, 21]. The theory of associative n-categories, which is the basis of homotopy.io, was first developed by Dorn, Douglas, and Vicary and described in Dorn's PhD thesis [12].

2 Using the Tool

The tool consists of two main components: the *signature* and the *workspace*. The signature stores a list of generators for an n-category, and the workspace stores an n-diagram in the free n-category generated by this signature. The tool implements a number of *actions* to modify the signature and/or workspace, and every state of the tool is determined by the list of actions that led to that state, starting from the empty signature and workspace. This makes it easy to implement an undo/redo system, and to reproduce a state by replaying the list of actions which is useful for debugging and testing.

All actions can be triggered by clicking on UI elements such as the buttons on the sidebar, interacting with the workspace diagram, or with keyboard shortcuts. We denote the keyboard shortcut associated to an action as A, which represents pressing the "A" key. Unlike a traditional proof assistant, there is no text-based aspect to the user-interface, except for metadata and for providing generator names. A screenshot of the user interface is shown in Figure 3. In this section, we describe its major components.

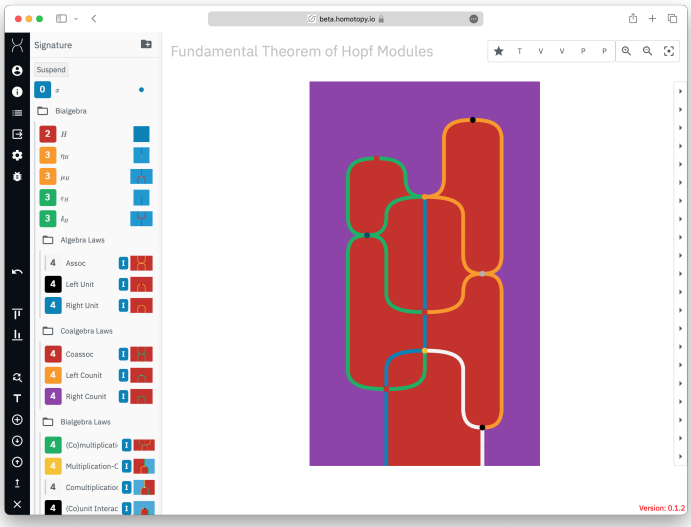


Figure 3 The interface of the proof assistant.

Adding generators. There are two primary ways to add new generators to the signature:

- We can add new 0-dimensional generators by clicking "Add 0-cell" (A) on the sidebar.
- We can add new $(n+1)$-dimensional generators by constructing a source n-diagram in the workspace and clicking "Source" (S), and then constructing a compatible target n-diagram in the workspace and clicking "Target" (T). The order of these two actions is not important, and the first diagram will be stashed and displayed in the bottom-left corner of the workspace until the second one is constructed.

Manipulating the workspace. Clicking on any generator in the signature will bring that diagram into the workspace. We can raise the dimension of the diagram with the "Identity" (I) action. We can construct composite diagrams by attaching another diagram to a boundary by clicking on the edge of the diagram. We can also use generators in the signature to perform rewrites by clicking *inside* the workspace diagram. Finally, we can perform homotopies, such as contractions and expansions which will be described in Section 5, by clicking and dragging.

Theorems. If the workspace is displaying an n-diagram D for $n > 1$, the "Theorem" (H) action becomes available. This action creates a new n-dimensional generator T with the same type as D, and a new invertible $(n+1)$-dimensional generator $P\colon T \to D$. This could be done by hand, and in this sense this action does not strictly add functionality, rather it adds a useful shortcut.

The idea is that T is an algebraic generator which axiomatises the existence of D, therefore allowing users to use it as a rewrite, and the generator P witness the fact that T is true (i.e. is inhabited), by rewriting it to D. This feature is useful for formalising complex proofs that depend on other lemmas, by saving each lemma as a theorem and then combining them to prove the main theorem. It can also be used to give definitions – i.e. we can think of T as a new generator which is *defined* to equal D, and P can then be used to expand the definition. This is similar to using the `abstract` keyword in Agda, to hide the "implementation detail" of a proof.

View control. Visualising n-diagrams for $n > 2$ presents a fundamental challenge, since the geometry can be difficult for us to visualise. A number of features are therefore provided for manipulating the visualisation: projecting out certain dimensions, navigating to a subdiagram, and changing the rendering dimension (with a choice between 0 and 4 dimensions).

These manipulations are effected via the *view control* component in the top-right corner of the workspace. This consists of a list of $n + 1$ buttons, where n is the dimension of the workspace diagram, corresponding to the following regular expression:

$$\bigstar\ (\mathsf{S} \mid \mathsf{T} \mid \mathsf{R}i \mid \mathsf{S}i)^k\ \mathsf{V}^d\ \mathsf{P}^{n-k-d} \qquad (d \leq 4)$$

Here S means source, T means target, Ri means the i-th regular height, and Si means the i-th singular height. This means that we are viewing a k-dimensional subdiagram of the workspace diagram, which is an $(n - k)$-diagram obtained by recursively going into heights as specified by the first k symbols. This subdiagram is then projected to d dimensions; that is, we are *viewing* d dimensions, and *projecting* the remaining $n - k - d$ dimensions. Clicking the star will reset the view to the original n-diagram (i.e. reset k to 0), and clicking on any of the k view buttons will reset the view to some prefix. Clicking on any V button will decrement d by 1, to a minimum of 0 (displayed as a single point) and clicking on any P button will increment d by 1, to a maximum of 4 (rendered as a movie of 3D geometries).

Users can descend into a height by clicking the chevrons appearing on the right-hand side of the diagram, or descend into the source height by pressing [→]. Similarly, while inside a height, users can ascend to the parent height by pressing [←], and navigate to adjacent k-th height components by pressing [↑]/[↓].

To illustrate how the projection works, consider Figure 1 which shows the 2D and 3D projections of a 3-dimensional diagram side-by-side. Note that the 2D projection can be understood as looking at the 3D projection from below, and projecting onto a 2D plane.

Generator options. When hovering over a generator, a ⚙ icon appears on the left which reveals a menu of options for that generator. This allows the user to rename the generator (with support for LaTeX) or change its colour or shape. Most importantly, it allows the user to mark a generator as invertible.

Image export. The sidebar has an "Image Export" button which reveals a panel for exporting the workspace diagram to different formats, such as SVG, TikZ, Manim, and STL.

3 Example: Eckmann-Hilton

Here we will illustrate how the proof assistant may be used in practice to formalise results in higher category theory. Our running example will be the Eckmann-Hilton argument, the key result in the correspondence between braided monoidal categories and 3-categories which are doubly-degenerate, meaning they have a unique 0-cell and no non-identity 1-cells. This will be essential in Appendix A for our formalisation of Hopf algebras in braided monoidal categories. This section is accompanied by a video tutorial [1]. The resulting workspace can be loaded into the tool at `https://beta.homotopy.io/p/2402.00002`.

To formalise the Eckmann-Hilton argument, we load the tool and begin constructing our signature, which is given by a unique 0-cell x and two 2-cells α, β which have source and target the identity on x. Since x is a 0-cell, we may immediately add this to the signature by pressing the "Add 0-Cell" button, and renaming the cell to "`$x$`" – the use of "`$`"s is optional, but indicates to the tool that the generator's name should be interpreted as LaTeX. To add

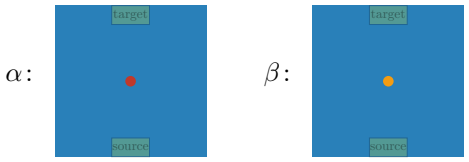


Figure 4 The scalars α and β in our signature for the Eckmann-Hilton argument.

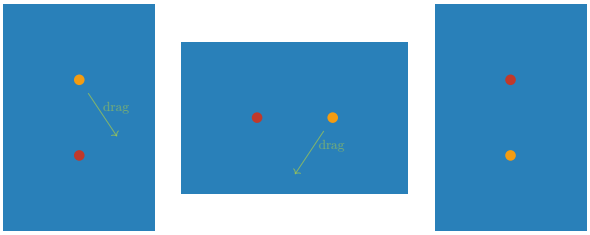


Figure 5 The Eckmann-Hilton argument.

the two scalars α and β to the signature, we need to provide the tool with their sources and targets, which in this case is 1_x. Selecting the generator x we have added to the signature, we make it the current working diagram. We then click on the "Identity" button to get 1_x from x, and take it as the source of a new 2-cell α by clicking on "Source". Repeating the construction of 1_x, we can provide the tool with the target of α. Since the diagrams have compatible boundaries, the globularity condition is satisfied and the tool allows to click on "Target" to create a 2-cell, which we rename to "`$\alpha$`". By repeating this procedure for β, we get the signature depicted in Figure 4.

With our signature in place, we can begin proving facts about the compositional behaviour of our scalars. The essence of the Eckmann-Hilton argument amounts to the commutativity and coincidence of vertical and horizontal composites of scalars, as depicted in Figure 5.

Let us detail how this proof can be built. We begin by constructing the right-most diagram in Figure 5 by selecting α from the signature, clicking on the highlighted region marked as *target* in Figure 4 and selecting β to attach. This builds the vertical composite of α and β and gives us the starting point or source for our proof. Since we wish to collect the steps of the proof into a 3-cell, we take the identity on the current diagram. This does not appear to change the current diagram displayed but adds an extra dimension in the view control.

We then drag β towards α along the right to get the middle diagram, triggering a *contraction* procedure, which we will detail in Section 5.2. Next, we drag β further downwards, triggering the dual procedure of *expansion*, detailed in Section 5.3. Note that these procedures only succeed if they are sound, i.e. the resulting diagram has a valid type as checked by the procedure of Section 5.4.

We now inspect the proof from the top-dimension by clicking on the $\star$ button on the right. From this view, our proof amounts to the construction of a braid as depicted in the first image of Figure 6. We can see this more clearly if we contract the middle part of the diagram by vertically dragging the bottom half-braid towards the top, obtaining the second image of Figure 6. We may also perform the proof entirely from this view, by horizontally dragging the legs of the braid past each other. A 3D visualisation of this proof can now be observed by pressing the P button in the view control.

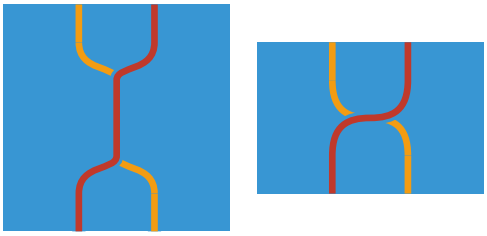


Figure 6 The Eckmann-Hilton proof (left) and its contraction (right).

4 Core Data Structures

In this section we outline the fundamental algebraic data types which are used to encode n-diagrams in the implementation. These are diagrams, rewrites, cones and cospans; their definitions are sketched in Figure 7. Note that the Rust type system, along with our choice of encoding, has the following limitation: there exist valid terms of the `diagram` type which do not correspond to admissible n-cells of the free higher category generated by the current signature. For example, any `Diagram0` which consists of a generator with its `dimension` field set to a non-zero integer does not correspond to any 0-cell of any higher category. The typechecking procedure of Section 5.2 is used to determine validity of these terms.

```
type frame = int
type generator = { dimension: int; id: int }

type rewrite =
  | Rewrite0Identity
  | Rewrite0 of { source: generator; target: generator; label: frame }
  | RewriteN of { cones: cone list }

and cone = {
  index: int;
  source: cospan list;
  target: cospan;
  slices: rewrite list;
}

and cospan = { forward: rewrite; backward: rewrite }

type diagram =
  | Diagram0 of generator
  | DiagramN of { source: diagram; cospans: cospan list }
```

Figure 7 The core data structures of homotopy.io.

4.1 Diagrams

The core data structure of homotopy.io is a recursive encoding for diagrams that is derived from the zigzag construction. A 0-diagram has trivial shape (a single point), and type is essentially the labelling which is an assignment of name to colour, which we call a *generator*. An $(n+1)$-diagram is determined by an alternating sequence of singular and regular heights, which are n-diagrams, together with information on how the heights fit together:

$$r_0 \xrightarrow{f_0} s_1 \xleftarrow{b_0} r_1 \xrightarrow{f_1} s_1 \xleftarrow{b_1} r_1 \xrightarrow{f_2} s_2 \xleftarrow{b_2} r_2.$$

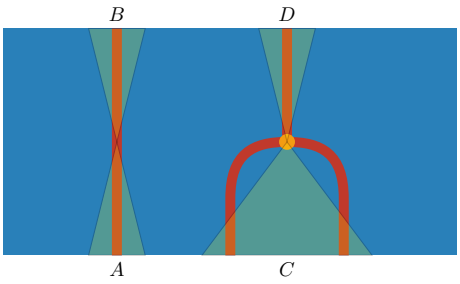
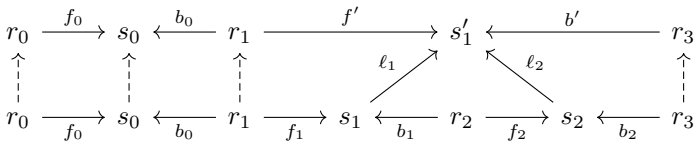


Figure 8 A zigzag with four cones. A and B are *identity* cones that do not change the diagram, omitted in the sparse representation.

Each singular height s_i is equipped with rewrites $f_i \colon r_i \to s_i$ and $b_i \colon r_{i+1} \to s_i$ that encode the difference between s_i and its neighbouring regular heights r_i and r_{i+1}. We call the pair (f_i, b_i) a cospan, and describe the height of a diagram as the number of cospans (equally, singular heights) it contains. There is always one more regular height than singular heights. Instead of storing all the heights of a diagram, we only keep around the first regular height r_0 and reconstruct the other heights when needed by applying the rewrites backwards and forwards.

4.2 Rewrites

Rewrites between diagrams are also encoded as a recursive data structure. A rewrite of 0-diagrams (0-rewrite) is either the identity rewrite or a rewrite between the underlying generators; in the latter case, it is also equipped with a *frame* which represents the directionality of the rewrite in some space associated to the ambient diagram. A $(n+1)$-rewrite $x \to y$ modifies the sequence of cospans in x by removing subsequences and replacing them with individual cospans. Each such modification is called a *cone* (see Figure 8). The following diagram, which encodes the bottom half of Figure 8[1], is an example of a rewrite with a single cone that replaces the cospans (f_1, b_1) and (f_2, b_2) with the cospan (f', b'):



A cone in a rewrite of $(n+1)$-diagrams also contains rewrites between the n-dimensional singular heights. In the example above these are the rewrites $\ell_1 \colon s_1 \to s'_1$ and $\ell_2 \colon s_2 \to s'_1$. A rewrite does not store information for the parts of the diagrams that do not change. Since in practice most adjacent heights only differ in small parts, this sparse encoding leads to significant space efficiency.

The core data structures admit a series of auxiliary algorithms:

1. Given an n-diagram x we can apply a rewrite $x \xrightarrow{R} y$ forwards to obtain y. Similarly, given an n-diagram y and a rewrite $x \xrightarrow{R} y$ we can apply R backwards to y.

[1] This can be seen by overlaying the diagram along the bottom of Figure 8. The identity map $s_0 \dashrightarrow s_0$ corresponds to cone A, and the right component consisting of s_1, r_2, s_2, s'_1 and maps between them corresponds to cone C.

2. We can compute the heights of an $(n+1)$-diagram at any singular or regular height. In particular, we can obtain the source and target of a diagram.
3. When $x \xrightarrow{R} y$ and $y \xrightarrow{R'} z$ are rewrites, we can compute the composite rewrite $x \xrightarrow{R' \circ R} z$ while preserving sparsity.
4. For two n-diagrams s and t whose sources and targets agree, we can create an $(n+1)$-dimensional diagram which represents an $(n+1)$-morphism between s and t.
5. Given an n-diagram x and $k \leq n$ we can compute the generators that would be visible in the projection of x to k dimensions.
6. For an n-diagram x and $k \leq n$ we can produce a graph that is a dense encoding of the projection of x to k dimensions. The simplicial complex obtained as the flag complex from this graph has the geometry of the rendered diagram. The layout algorithm then assigns coordinates to the vertices (see Section 5.5).
7. Given a pair of diagrams x, y we can search for copies of y that are embedded into x and intersect a line through the projection. This allows us to find opportunities to rewrite a diagram, which itself can be realised as a higher cell.

5 Key Algorithms

Many of our key algorithms described in this section first compute the shape of the result, usually by recursion over diagram dimension, and then complete the type information. These data are encoded by directed graphs, for which we can utilise graph algorithms to implement our operations.

5.1 Collapse

A string diagram has geometric properties (e.g. the length of a wire) which are not intended to be meaningful, as its meaning is captured entirely by topological properties (e.g. connectivity of a wire). Topology yields a natural compatibility with composition in the string diagram calculus: if two wires are placed in sequence, then the resulting string diagram is merely a longer wire, and therefore it looks topologically the same and represents the same mathematical content. However, both the tool and the theory are based on *combinatorial* encodings of string diagrams, and moreover there is a distinct encoding for the diagram of two wires in sequence versus one wire.

We describe the *collapse* of a diagram as a combinatorial representation of these topological invariants, essentially as a directed graph, which gives a normal form for the fully-exploded graph obtained from an n-diagram in a frame-preserving way. In more detail, given such a graph, every node has a neighbourhood which determines a set of outgoing frames and incoming frames (framing data); now, consider the largest equivalence relation on nodes determined by $x \sim y$ when x is adjacent to y by an identity 0-rewrite, and x and y admit equal framing data; collapse is the quotient graph induced by $\sim$. Explicitly, we compute this by treating the graph as a simplicial complex and checking each 1-simplex (edge) which is an identity 0-rewrite to see if it is collapsible, whereby identifying both 0-simplex faces (endpoints) respects $\sim$, by checking all 2-simplices (triangles) for which that 1-simplex is a face. A simple example of this is given in Figure 9.

Collapse is used to compare when two diagrams differ as encodings but have the same topological data: many different diagrams may have the same collapse, for instance degenerating a diagram along any part[2] does not change its collapse, but two diagrams with distinct

[2] This corresponds to composition with a weak unit in the n-categorical model; combinatorially, this represents the insertion of redundant data.

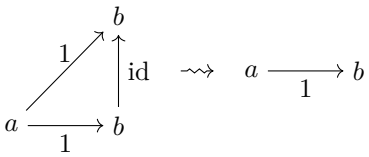


Figure 9 The edge $b \xrightarrow{\mathrm{id}} b$ is collapsible, so the graph can be simplified in a frame-preserving fashion.

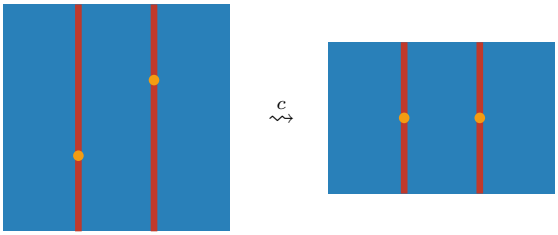


Figure 10 A contraction of two beads on two singular heights.

collapses are necessarily distinct. These degeneracies arise naturally in the course of the other operations described in this section, but can be often eliminated without rendering the diagram malformed. Collapse is a crucial component of typechecking, analogously to how the standard technique of deciding equality in a term calculus often amounts to computing normal forms for each term. A detailed account is given in [21].

5.2 Contraction

Contraction is the process of shortening a diagram along a homotopy equivalence which reduces the height of a diagram locally, producing a contraction rewrite $D \xrightarrow{c} C$, which may or may not exist depending on D. An example of this is given by Figure 10, which combines both singular heights on the left into one on the right. The result of the contraction C always has a singular height of one. Informally, as a homotopy, the contraction rewrite c is *the* unique canonical way to shorten the diagram without making any arbitrary choices, and this is mathematically captured by its description as the computation of a categorical colimit.

Contraction works recursively on diagram dimension, with the 0-dimensional base case obtained via collapse, as in Section 5.1, and then ensuring that the maximal elements of the poset induced by the reachability relation on the resulting directed graph are compatible [21]. The higher-dimensional recursive case has been described theoretically by Reutter and Vicary [33]; it works by first determining a "Δ-colimit", which determines the shape that the result necessarily possesses, and then if that exists it then attempts to find a compatible labelling of each stratum to complete the typing information by a divide-and-conquer strategy.

In the base case, compatibility is the uniqueness of the labelling of the maximal element of the poset, combined with the condition that the framing data on each maximal element is identical. In the recursive case, for an $(n+1)$-rewrite $D \xrightarrow{c} C$, the algorithm is as follows:

1. D is associated to a directed graph G whose nodes represent regular and singular heights of D, weighted by n-diagrams, and whose edges form cospans between regular heights and are weighted by n-rewrites;
2. this directed graph is then exploded, obtaining a larger directed graph E whose nodes are weighted by $(n-1)$-diagrams, and whose edges are weighted by $(n-1)$-rewrites, by replacing each node of G with a directed graph, as in the previous step, and each edge by a collection of connecting $(n-1)$-rewrites;

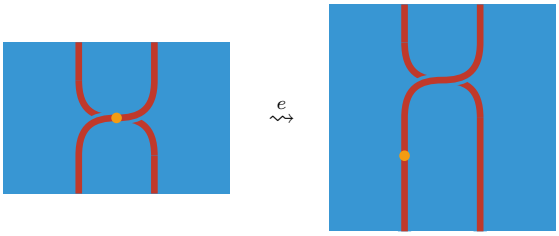


Figure 11 An anticontraction move.

3. E induces a subgraph Δ_E which represents only the shape data of D consisting of only the singular-singular nodes; on Δ_E, we obtain its condensation graph with respect to strongly-connected components[3]; if the edge relation of the condensation graph does not correspond to a linear order on nodes, then fail, otherwise, this determines:
   a. subproblems determined by the subgraph of E induced by reachability for the nodes of a particular strongly-connected component of Δ_E;
   b. a linear ordering of subproblems, which will determine how their solutions should be combined.
4. each subproblem represents an independent fragment of D which spans its entire height, and is solved recursively (step 2); its solution is a singular-singular height of C, which necessitates the linear ordering of subproblems; the rest of C and c are constructed from the remaining data;
5. the result is then typechecked, and if it fails, then the algorithm fails.

There are also auxiliary algorithms which propagate a contraction within some height of a larger diagram. We refer the interested reader to [33].

5.3 Expansion

Expansion is the dual of contraction, first described by Reutter and Vicary [33], and later refined by Tataru and Vicary [40]. It takes a diagram D and produces an expanded diagram E, such that E in fact contracts to give D via some rewrite $E \xrightarrow{c} D$. In this sense, expansion is a partial converse to contraction.

It is defined inductively, similarly to contraction. In the base case, it performs an interchanger move that separates two singular levels (the reverse of Figure 10). In the recursive case, expansion attempts to propagate an expansion of a sub-diagram to the diagram itself, yielding a diagram which will contract to the original. Since contraction is computed by a colimit process, this requires an algorithm that can reverse the ordinary colimit process, a procedure that we call *anticontraction* [40]. We illustrate this with the recursive expansion example in Figure 11, where a vertex is moved out of a singular height. Note that the expanded diagram on the right indeed contracts to give the original diagram on the left.

[3] This graph has a node for each strongly-connected component, and each edge represents reachability for strongly-connected components.

Figure 12 The cancellation 2-cell $f^{-1} \circ f \to 1_x$ generated by an invertible 1-cell $f : x \to y$.

5.4 Typechecking

Typechecking is the process of checking the validity of a diagram with respect to a given signature. Note that despite the name, in our context, "typechecking" does not refer to a type-theoretic procedure determined by inference rules. The underlying theory gives a clear perspective on this process [12, 17]: break the diagram into atomic "pieces", and ensure that each piece collapses to the canonical neighbourhood of the respective generator. This tells us that the neighbourhood of each point in the diagram is fully described by the signature.

Collapse plays a crucial role here, because it ensures that higher-dimensional coherences of invertible generators also typecheck. For example, if $f : x \to y$ is an invertible 1-cell, we can generate a 2-cell $f^{-1} \circ f \to 1_x$ known as the *counit* (see Figure 12); this is well-typed because it collapses to the canonical neighbourhood of f.

5.5 Layout

The layout algorithm is used to assign coordinates to every point in an n-diagram, and is a crucial component of the rendering pipeline described in Section 6.2, enabling rendering in 2D, 3D and 4D. We use a categorical construction based on factorisation systems and colimits to extract a set of linear constraints from the total order data of the diagram [39]. These constraints encode the necessary conditions for a layout to be well-defined. For example, in 2D, this would include the information that one wire is to the left of another wire; or, in 3D, that one surface is in front of another. We further impose aesthetic constraints, e.g. that wires and surfaces should be centred. Finally, these constraints are passed to the linear solver HiGHS [22] to find a layout that satisfies all constraints. An example layout for a 2-diagram is given in Figure 13: each node receives a cartesian coordinate (x, y) from the solver, which is used to determine where it should be rendered on-screen.

6 Implementation

Here, we describe further implementation details: the memoisation techniques used in our core data structures and aspects of the rendering pipeline. Whilst independent of the tool's mathematical foundations, we have found many of the details here essential, particularly in enabling the tool to operate in a resource-constrained environment – i.e. the browser.

6.1 Memoisation

Our data structures for `Diagram` and `Rewrite` are immutable. Operations that would modify an object instead create a new one with the modifications applied. This allows us to apply a technique known as *hash consing* [14]: whenever a new `Diagram` or `Rewrite` is created, we

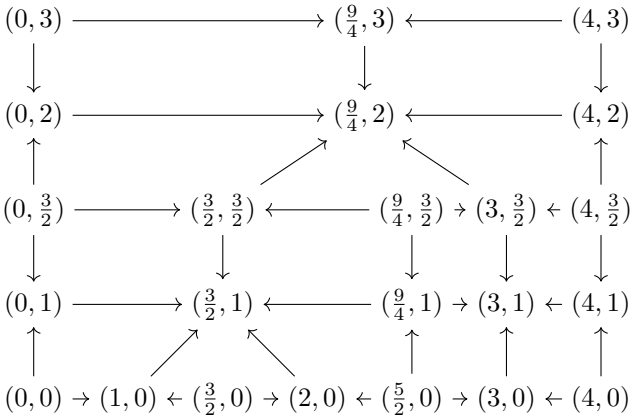


Figure 13 The layout of a 2-diagram.

check in a hash table if a logically equal instance is already present and reuse it if possible. Hash consing enforces the invariant that any two instances that are logically equal must become physically equal – i.e. have the same representation and location in memory. To test for equality we may therefore perform a simple pointer comparison instead of traversing the entire deeply nested structure; similarly, we cache the hash value of every `Diagram` and `Rewrite` to avoid deep traversals. We incur a performance cost due to the hash table lookup involved whenever a new instance is created. However, we observe significant improvements in performance and memory usage overall. In practice, we have found that even under the sparse encoding of Section 4, directly representing diagrams remains highly redundant: applying hash consing makes memory usage almost negligible, even for large diagrams.

The algorithms from Section 5 are largely recursive over the structure of a diagram and perform many repeated recursive calls on logically equal substructures. We therefore memoise the results of recursive calls. Since our deduplicated representation allows for very fast equality checks and caches hash values, lookups in the memoisation table are comparatively cheap.

Our data structures are reference counted, so we know when the last reference to an instance goes out of scope and it is safe to remove it from the hash table. The recursive algorithms operating on diagrams temporarily materialise the structure that is implicit in the sparse encoding, leading to the same `Diagram` or `Rewrite` being created and destroyed many times during the execution of the algorithm. An eager approach to maintaining the deduplication hash table therefore leads to unnecessary churn. We therefore delay the removal of dead objects from the hash table to a batched garbage collection step that walks the table and removes instances that have no remaining references.

6.2 Rendering Pipeline

One of the most important parts of the implementation is the rendering pipeline, which allows for visualising n-diagrams in up to four dimensions. It consists of three components. First, we have the *layout* algorithm described in Section 5.5 that assigns real coordinates to every point of an n-diagram. Second, there is the *mesh generation*, which takes an n-diagram and computes a cubical mesh – i.e. a subdivision of the n-diagram into *abstract* k-dimensional cubes for $k \leq n$. Together, these data are called a *geometry* which is a collection of k-dimensional cuboids in the Euclidean space $\mathbb{R}^n$. Finally, we have a *subdivision*

procedure which takes a geometry and makes it smoother by recursively subdividing each cube into smaller cubes with the positions of the new vertices calculated by interpolating the positions of the old vertices.

The subdivided geometry can then be sent to a number of rendering engines. Amongst these, the most important are the SVG renderer, rendering 2D diagrams for all the user's primary interaction, and the WebGL renderer, rendering 3D and 4D diagrams (with the 4D diagrams appearing as *smooth* animations of 3D diagrams). We also support a TikZ renderer, which can be used to generate high quality string diagrams for use in papers such as in this article.

In the 4-dimensional case, we obtain the animation by intersecting the geometry with an axis-aligned hyper-plane. This computation is easily implemented via a compute shader. However, to maximise homotopy.io's compatibility with contemporary browsers, our shading pipeline is implemented in WebGL2, precluding the use of compute shaders.

This situation presents a challenge, as computing intersection geometries offline for large diagrams is prohibitively expensive in terms of memory consumption. Thus, we are forced perform the slicing in real-time, using a subtle rendering trick:

1. each cube is broken down into a disjoint collection of covering simplices (Figure 14, left);
2. each simplex is further decomposed into sub-simplices, each with one face whose vertices have equal w-components[4],
3. of the six edges of each simplex, the three with non-zero w-components are converted to three GL vertices, and the simplex a corresponding polygon (Figure 14, right);
4. each GL vertex is supplied with the start and end coordinates of the corresponding edge;
5. the vertex shader interpolates between these points given a global w-coordinate, passed in as a uniform, dropping the vertex whenever w is outside the bounds of the edge.

The effect is that the animation is broken down into a collection of "birth" and "death" events for small polygons, which can be rendered straightforwardly, as if it were a 3D mesh. Animated wires in 4-dimensions are produced through a similar process, but the "pipe"-like appearance is produced via a deferred post-processing effect, to minimise the cost of storing geometry for the cylinders.

We additionally compute normals for intersection geometries in real-time via a barycentric interpolation across a volume-weighted average taken at each vertex. This is directly analogous to the calculation of normals in traditional 3D pipelines. However, much care is required when it comes to calculating and preserving the orientations of the elements of the mesh.

7 Future work

The tool is under active development, both mathematically as an end itself and as a piece of research software as a means to facilitate the development of higher-categorical mathematics. We sketch some ideas for future improvements to the tool:

Functor boxes It would be both interesting and practical to consider what a functor would mean in this context; graphically, it should allow for "boxing" pieces of diagrams, allowing those components to be treated as atomic; this would present the user with more flexibility in representing proofs.

[4] w-component refers to the fourth coordinate of the homogeneous coordinate system commonly used in computer graphics, and in our tool via WebGL.

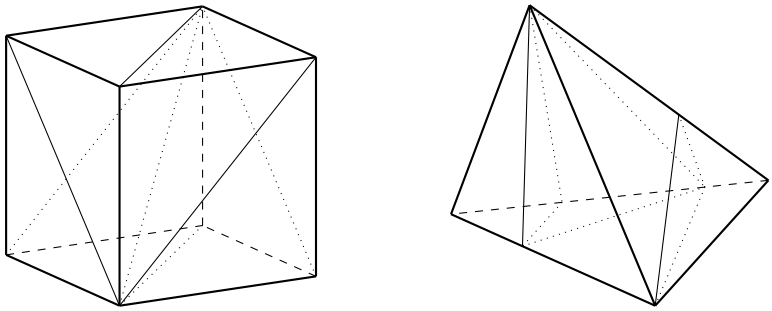


Figure 14 The decomposition of a 3-cube into six 3-simplices (left) and a 3-simplex into w-aligned sub-simplices (right).

Dualisability support The next logical step after the introduction of coherent inverses would be the introduction of coherent duals, which would allow for a mathematical exploration of important problems in higher algebra, such as the tangle hypothesis [4], and knot theory.

Interopability with type theory Some arguments of logical flavour are more easily expressed type-theoretically, e.g. in homotopy type theory or CaTT [15], while more graphical arguments are better suited to homotopy.io; ideally, we could make these tools talk to each other to support bimodal reasoning.

Linearity Support for higher linear categories would make the tool amenable to directly performing calculations in topological quantum field theory.

Gallery An arXiv-like gallery can be presented to showcase published proofs, better enabling proof discovery.

Cubical version More general notions of string diagrams for cubical n-categories have been considered [32] and it would be an interesting question to extend homotopy.io to cubical shapes.

References

1 Constructing a braiding in homotopy.io. URL: `https://homotopy.io/braiding-example`.

2 Nicolás Andruskiewitsch and Walter Ferrer Santos. The Beginnings of the Theory of Hopf Algebras. *Acta Applicandae Mathematicae*, 108(1):3–17, October 2009. `doi:10.1007/s10440-008-9393-1`.

3 Michael F Atiyah. Topological quantum field theory. *Publications Mathématiques de l'IHÉS*, 68:175–186, 1988.

4 John C Baez and James Dolan. Higher-dimensional algebra and topological quantum field theory. *Journal of mathematical physics*, 36(11):6073–6105, 1995.

5 John C Baez and Martin Neuchl. Higher dimensional algebra: I. braided monoidal 2-categories. *Advances in Mathematics*, 121(2):196–244, 1996.

6 Krzysztof Bar, Aleks Kissinger, and Jamie Vicary. Globular: an online proof assistant for higher-dimensional rewriting. *Logical Methods in Computer Science*, 14, 2018. `doi:10.23638/LMCS-14(1:8)2018`.

7 John W Barrett, Catherine Meusburger, and Gregor Schaumann. Gray categories with duals and their diagrams. *arXiv preprint*, 2012. `arXiv:1211.0529`.

8 Yuri Bespalov and Bernhard Drabant. Hopf (bi-)modules and crossed modules in braided monoidal categories. *Journal of Pure and Applied Algebra*, 123(1-3):105–129, 1998.

9 Simon Burton. String diagrams for higher mathematics with wiggle.py, 2023. Talk at SYCO 11. URL: `https://arrowtheory.com/wiggle_demo.pdf`.

10 Giovanni de Felice, Alexis Toumi, and Bob Coecke. Discopy: Monoidal categories in python. *arXiv preprint*, 2020. arXiv:2005.02975.

11 Antonin Delpeuch. A complete language for faceted dataflow programs. *arXiv preprint*, 2019. arXiv:1906.05937.

12 Christoph Dorn. Associative n-categories. *arXiv preprint*, 2018. arXiv:1812.10586.

13 Christoph Dorn and Christopher L Douglas. Manifold diagrams and tame tangles. *arXiv preprint arXiv:2208.13758*, 2022.

14 Jean-Christophe Filliâtre and Sylvain Conchon. Type-safe modular hash-consing. In *Proceedings of the 2006 Workshop on ML*, pages 12–19, 2006. doi:10.1145/1159876.1159880.

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

16 Amar Hadzihasanovic and Diana Kessler. Data structures for topologically sound higher-dimensional diagram rewriting. *arXiv preprint*, 2022. arXiv:2209.09509.

17 Lukas Heidemann, David Reutter, and Jamie Vicary. Zigzag normalisation for associative n-categories. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 1–13, 2022. doi:10.1145/3531130.3533352.

18 Martin Hofmann and Thomas Streicher. The groupoid interpretation of type theory. *Twenty-five years of constructive type theory (Venice, 1995)*, 36:83–111, 1998.

19 Heinz Hopf. Uber Die Topologie der Gruppen-Mannigfaltigkeiten und Ihre Verallgemeinerungen. *Annals of Mathematics*, 42(1):22–52, 1941. Publisher: Annals of Mathematics. doi:10.2307/1968985.

20 Nick Hu, Alex Rice, and Calin Tataru. sd-visualiser, 2024. URL: https://github.com/sd-visualiser/sd-visualiser.

21 Nick Hu, Calin Tataru, and Jamie Vicary. Coherent invertibility in associative n-categories. In preparation, 2024.

22 Qi Huangfu and JA Julian Hall. Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10(1):119–142, 2018. doi:10.1007/s12532-017-0130-5.

23 André Joyal and Ross Street. The geometry of tensor calculus, i. *Advances in mathematics*, 88(1):55–112, 1991.

24 Aleks Kissinger and Vladimir Zamdzhiev. Quantomatic: A proof assistant for diagrammatic reasoning. In *Automated Deduction-CADE-25: 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings 25*, pages 326–336. Springer, 2015. doi:10.1007/978-3-319-21401-6_22.

25 Richard Gustavus Larson and Moss Eisenberg Sweedler. An Associative Orthogonal Bilinear Form for Hopf Algebras. *American Journal of Mathematics*, 91(1):75, January 1969. doi:10.2307/2373270.

26 Tom Leinster. *Higher operads, higher categories*. Number 298 in London Mathematical Society Lecture Note Series. Cambridge University Press, 2004.

27 Jacob Lurie. *Higher topos theory*. Princeton University Press, 2009.

28 V. Lyubashenko. Modular transformations for tensor categories. *Journal of Pure and Applied Algebra*, 98(3):279–327, February 1995. doi:10.1016/0022-4049(94)00045-K.

29 S. Majid. Algebras and Hopf Algebras in Braided Categories, September 1995. arXiv:q-alg/9509023. arXiv:q-alg/9509023.

30 Shahn Majid. Braided groups and algebraic quantum field theories. *letters in mathematical physics*, 22:167–175, 1991.

31 John W. Milnor and John C. Moore. On the Structure of Hopf Algebras. *The Annals of Mathematics*, 81(2):211, March 1965. doi:10.2307/1970615.

32 David Jaz Myers. String diagrams for double categories and equipments. *arXiv preprint*, 2016. arXiv:1612.02762.

33 David Reutter and Jamie Vicary. High-level methods for homotopy construction in associative n-categories. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13. IEEE, 2019. doi:10.1109/LICS.2019.8785895.

34 Emily Riehl and Dominic Verity. *Elements of ∞-Category Theory*, volume 194. Cambridge University Press, 2022.

35 Chiara Sarti and Jamie Vicary. Posetal diagrams for logically-structured semistrict higher categories. *arXiv preprint*, 2023. arXiv:2305.11637.

36 Christopher John Schommer-Pries. *The classification of two-dimensional extended topological field theories*. University of California, Berkeley, 2009.

37 Pawel Sobocinski, Paul W Wilson, and Fabio Zanasi. Cartographer: A tool for string diagrammatic reasoning (tool paper). In *8th Conference on Algebra and Coalgebra in Computer Science (CALCO 2019)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.

38 Moss E. Sweedler. *Hopf algebras*. Mathematics lecture note series. Benjamin, New York, NY, 1969.

39 Calin Tataru and Jamie Vicary. A layout algorithm for higher-dimensional string diagrams. *arXiv preprint*, 2023. arXiv:2305.06938.

40 Calin Tataru and Jamie Vicary. The theory and applications of anticolimits. *arXiv preprint*, 2024. arXiv:2401.17076.

41 The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. https://homotopytypetheory.org/book, Institute for Advanced Study, 2013.

42 Shouchuan Zhang. Braided hopf algebras. *arXiv preprint*, 2005. arXiv:math/0511251.

A Case Study: Hopf Algebras and Hopf Modules

In this appendix, we will detail an extended case study to highlight how homotopy.io may be practically employed to formalise mathematical results of substantial character. The basis we will build on for this case study will be the discussion in Section 3, showing that the Eckmann-Hilton argument could be used to formalise braids in the tool. This will allow us to define Hopf algebras and Hopf modules in braided monoidal categories, and build towards a proof of the *fundamental theorem of Hopf modules*, asserting the freeness of Hopf modules when certain idempotents split. Since this formalisation will involve rich algebraic gadgets, long proofs and universal properties, we will build it up gradually. The final formalisation, consisting of the full proofs, is available at https://beta.homotopy.io/p/2402.00006 so that the reader can inspect the proofs in detail and visualise them in 3D and 4D.

Hopf algebras originated in work by Hopf [19] in topology, developed on and popularised by Milnor and Moore [31] (see [2] for more detail). They have served to generalise results from group theory to objects of study in topology, knot theory, algebraic geometry, combinatorics and quantum theory [38]. Our discussion here is informed by the original proof of the fundamental theorem of Hopf modules for Hopf algebras over principal ideal domains is given in [25], its generalisation to braided monoidal categories in [28] and the alternative proof using Karoubi completions given in [8]. We also use the diagrammatic calculus for Hopf modules introduced by Majid in [30], which has been used to give a string diagrammatic translation of this result in [42], though the reader should be warned that the proof contains flaws.

We start by first defining a bialgebra H in a braided monoidal category. Recall from Section 3 that in order to obtain the braiding, we will start by adding a single 0-cell x to our signature, and then add our 2-cell H by specifying 1_x as source and target. Hence, H is an object in our braided monoidal category. Moreover, we need to add four 3-cells: a unit $\eta\colon 1 \to H$, a multiplication $\mu : H \otimes H \to H$, a counit $\varepsilon\colon H \to 1$, and a comultiplication $\delta : H \to H \otimes H$, which are depicted in Figure 15.

For H to be an *algebra*, η and μ must satisfy associativity and unitality laws, which are witnessed by the 4-cells depicted in Figure 16. We add these to the signature and mark them as invertible. For H to be a *coalgebra*, it will have to satisfy dual laws of coassociativity and counitality. Finally, for H to be a *bialgebra*, the algebra and coalgebra structure have to interact: units must propagate through comultiplications, counits through multiplication, counit and units must be one-sided inverses, and multiplication and comultiplication have to interact via the braiding. These interaction laws are specified as in Figure 17.

Thus, H is now a bialgebra. In order for it to be a *Hopf algebra*, we must additionally equip it with an antipode map $\sigma\colon H \to H$. This map needs to satisfy the two cancellation laws given in Figure 18, which make it an inverse to the identity map on H under the convolution product. These two laws are enough to establish the following lemmas on the interactions between the antipode and the bialgebra structure.

▶ **Lemma 1.** *For a Hopf algebra H in a braided monoidal category, we have $\sigma \circ \eta = \eta$.*

▶ **Lemma 2** ([29, Lemma 2.3])**.** *For a Hopf algebra H in a braided monoidal category, we have $(\sigma \otimes \sigma) \circ \gamma_{H,H} \circ \delta = \delta \circ \sigma$, where $\gamma_{H,H}\colon H \otimes H \to H \otimes H$ is the braiding.*

Proof. The essential steps of the graphical proof are given in Figure 25. We start by constructing the left-hand side of the equation, as it involves a braid and thus is more complex than the right-hand side. This gradient of complexity helps us to use the contraction procedure effectively.

The first step of the proof is to introduce units at the top of the diagram, and a counit at the bottom. We then braid the right-most unit past the antipode on the left, multiply the units and use them with the counit to introduce an antipode on the left. Above this antipode, we then use the multiplication/comultiplication axiom. Here we note that for proof to go through, the orientation of this braid, the braid in the starting diagram and the braid in the multiplication/comultiplication interaction axioms must match.

A sequence of (co)associativity and braid naturality moves follows, until we can isolate the right-most antipode into a bubble to eliminate it into a unit/counit pair, which we use to remove a pair of (co)multiplications. We then move to use the (co)associativity laws to isolate the right-most antipode into a bubble, and proceed to simplify the remaining diagram down to our target diagram by cancelling the (co)units. ◀

At this point, our signature contains our Hopf algebra H and two results about its properties, namely Lemma 1 and Lemma 2. Here we must note that the axioms for a Hopf algebra are self-dual, hence whenever we establish a result its dual statement, obtained by swapping unit and counit, multiplication and comultiplication and vice versa, will also hold. For instance, the dual of Lemma 1 would show that $\varepsilon \circ \sigma = \varepsilon$.

We will now add another object M with the same boundary as H, which is to be a *(left) H-Hopf module*. This means M is equipped with two maps, a (left) action $\alpha\colon H \otimes M \to M$ and a (left) coaction $\phi\colon M \to H \otimes M$, which are compatible with the structure of H. Thus, we add invertible cells as in Figure 20 to impose interaction between the multiplication μ and the action α, between the unit η and α, their duals for δ, ε and ϕ, and finally the interaction between the action/coaction pair.

Figure 15 The 3-cells η, μ, ε, and δ (left to right) in the bialgebra signature.

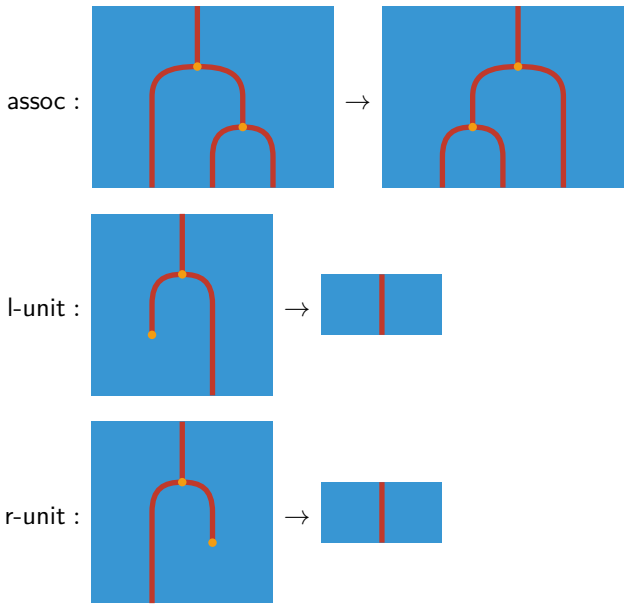


Figure 16 The associativity and unitality laws for the algebra (H, η, μ).

We now have an H-Hopf module M which we have manually added to the signature. The structure of the Hopf algebra H gives us another way to obtain H-Hopf modules: given any object B in our braided monoidal category, we may equip $H \otimes B$ with the structure of an H-Hopf module by taking the action to be $\mu \otimes 1_B \colon H \otimes H \otimes B \to H \otimes B$ and the coaction to be $\delta \otimes 1_B \colon H \otimes B \to H \otimes H \otimes B$. We call Hopf modules of this form *free*.

With the action and coaction of M and the antipode of H, we may define the endomorphism $\nu \coloneqq \alpha \circ (\sigma \otimes 1_M) \circ \phi \colon M \to M$ depicted in Figure 21. It turns out that this map is idempotent.

▶ **Lemma 3** ([8, Prop. 3.2.1]). *The map $\nu \colon M \to M$ is idempotent – i.e. $\nu \circ \nu = \nu$.*

Proof. We first use Lemma 2 to establish that $\phi \circ \nu = (\eta \otimes 1_M) \circ \nu$. This is given in Figure 22, where the key step is to use the action/coaction interaction axiom in order to apply Lemma 3, and then simplify down the diagram. With this at hand, idempotency follows from Lemma 1 and routine simplifications. ◀

With Lemma 3 at hand, we assume now that we have a splitting for ν – i.e. an object M^H and maps $\iota \colon M^H \to M$, $\upsilon \colon M \to M^H$ such that $\upsilon \circ \iota = 1_{M^H}$ and $\iota \circ \upsilon = \nu$. It turns out that this splitting enjoys an additional universal property.

▶ **Lemma 4** ([8, Prop. 3.2.1]). *The map $\iota \colon M^H \to M$ is the equaliser of ϕ and $\eta \otimes 1_M$.*

Proof. We add an object P and a map $\chi \colon P \to M$ such that $\phi \circ \chi = \eta \otimes \chi$ to the signature. We then show that $\chi = \nu \circ \chi$ by applying Lemma 1. But this means $\chi = \iota \circ (\upsilon \circ \chi)$ and since ι is monic this factorisation must be unique. ◀

Thus, M^H enjoys a universal property which trivialises the coaction, we call it *object of coinvariants* of M. Note that by dualising Lemma 4 and its dependencies, we may also prove

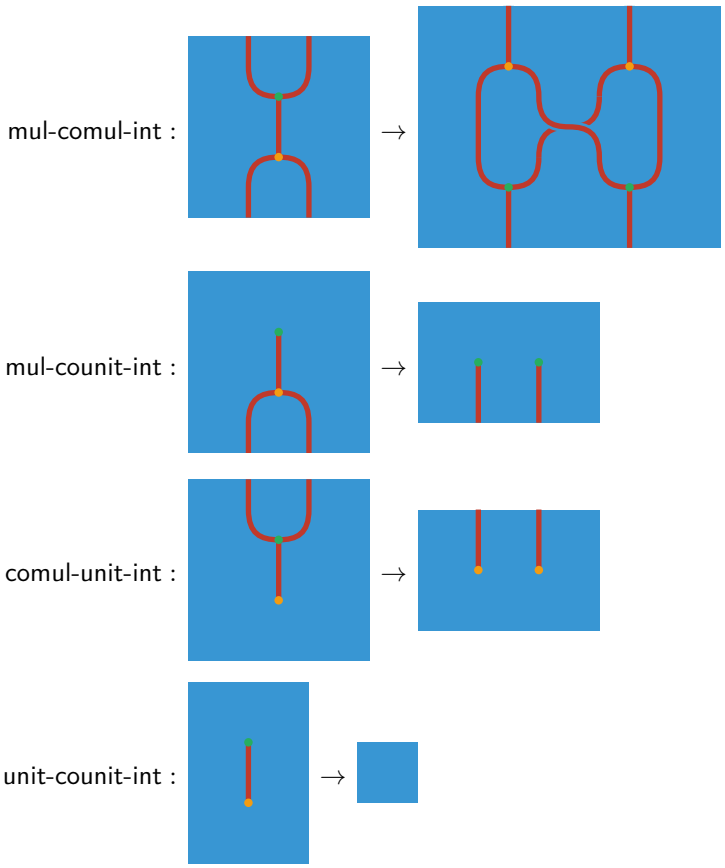


Figure 17 The bialgebra interaction laws for $(H, \eta, \mu, \varepsilon, \delta)$.

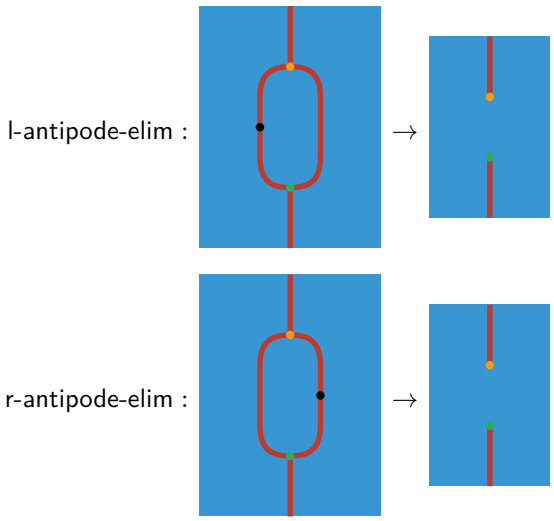


Figure 18 The antipode cancellation laws for $\sigma\colon H \to H$.

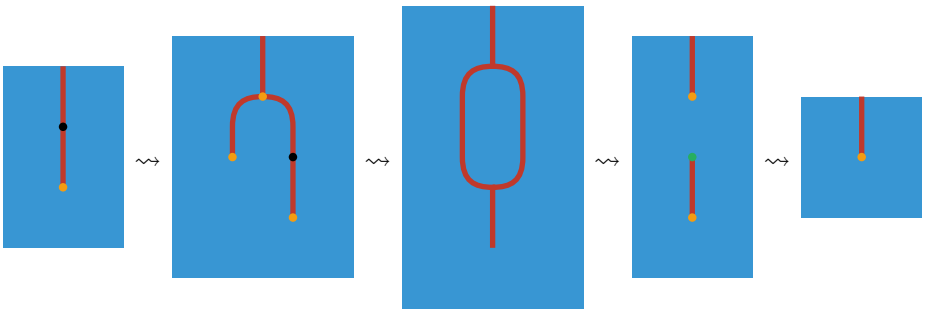


Figure 19 The proof of Lemma 1.

that υ is the coequaliser of α and $\varepsilon \otimes 1_M$, and thus M^H is also the *object of invariants* of M. Now the fundamental theorem of Hopf modules then asserts that if idempotents split in a braided monoidal category, up to isomorphism, every Hopf module is free.

▶ **Theorem 5** (Fundamental Theorem of Hopf Modules [25, Prop. 1],[8, Lemma 3.3.3])**.** *Let M be a left H-Hopf module in a braided monoidal category, and assume the map $\nu \colon M \to M$ splits through M^H. There is an isomorphism of left H-Hopf modules $H \otimes M^H \cong M$.*

Proof. We use the data of the splitting $\nu = \iota \circ \upsilon$ to construct two comparison maps $\xi \coloneqq \alpha \circ (1_H \otimes \iota)$ and $\psi \coloneqq (1_H \otimes \upsilon) \circ \phi$ between $H \otimes M^H$ and M, which will show induce the isomorphism of Hopf modules. The string diagrams corresponding to ξ and ψ are given in Figure 23.

We claim ξ respects the Hopf module structure. Compatibility between ξ and the actions α and $\mu \otimes 1_{M^H}$ follows immediately from the multiplication/action interaction law. As for the coactions, we follow Figure 24 in using the action/coaction interaction law, followed by the equaliser property given by Lemma 4. This establishes the claim.

Finally, we prove that ξ and ψ are mutual inverses. The proof of $\xi \circ \psi = 1_M$ is simpler, consisting mainly in the cancellation of the antipode, and is given in Figure 26. The proof of $\psi \circ \xi = 1_{H \otimes M^H}$ is more involved, making use of ν being idempotent (Lemma 3) as well as ι being an equaliser (Lemma 4). This is given in Figure 27. ◀

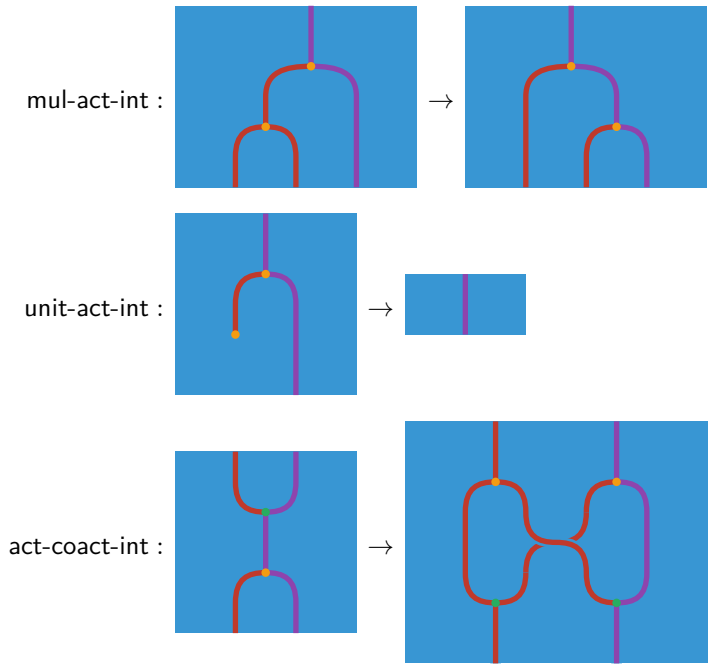


Figure 20 The H-Hopf module laws for M, with the duals to the first two omitted.

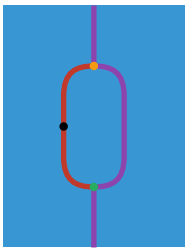


Figure 21 The idempotent endomorphism $\nu\colon M \to M$.

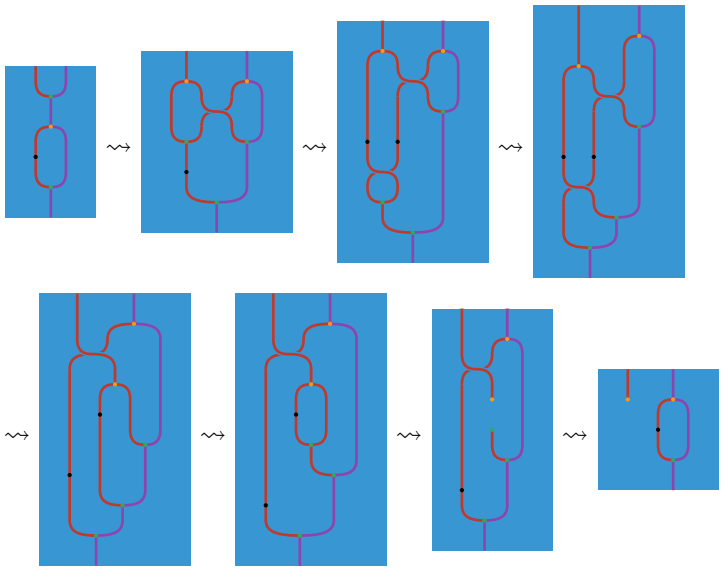
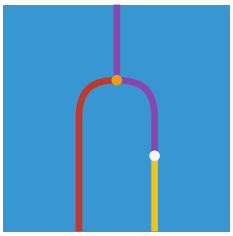
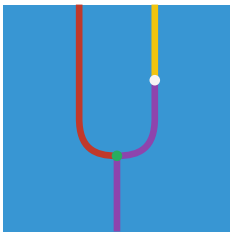


Figure 22 The proof of $\phi \circ \nu = (\eta \otimes 1_M) \circ \nu$ in Lemma 3.



(a) The map $\xi \colon H \otimes M^H \to M$.



(b) The map $\psi \colon M \to H \otimes M^H$.

Figure 23 The string diagrams for the mutually inverse ξ and ψ.

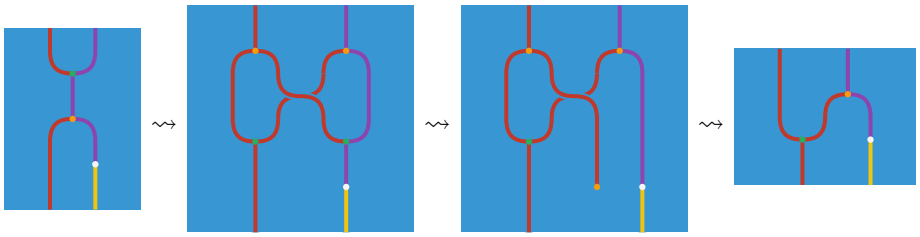


Figure 24 The map ξ respects the coactions ϕ and $\delta \otimes 1_{M^H}$.

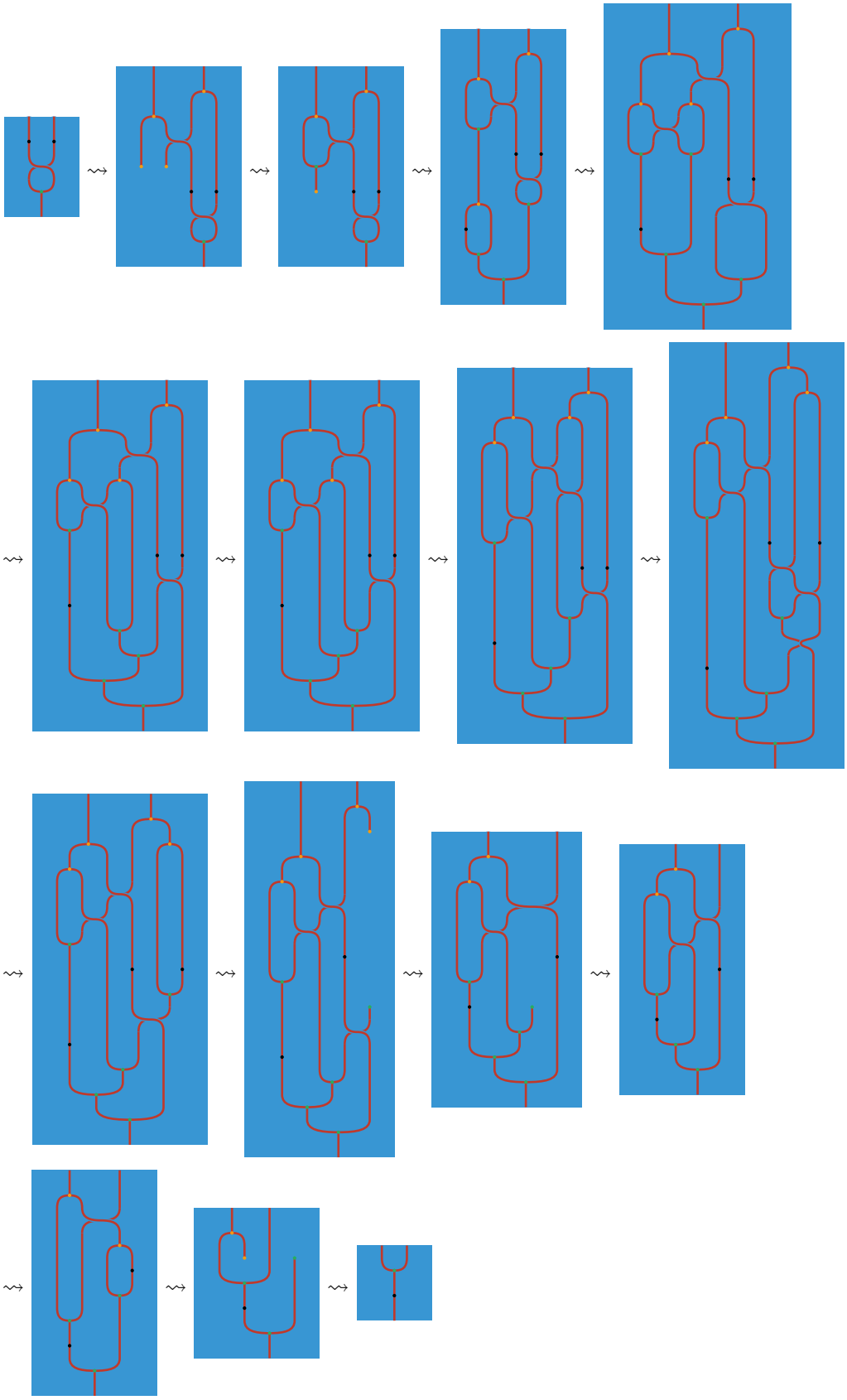


Figure 25 The proof of Lemma 2.

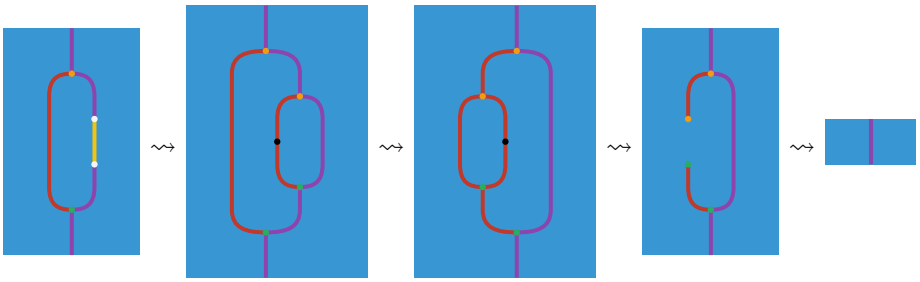


Figure 26 The proof of $\xi \circ \psi = 1_M$ in Theorem 5.

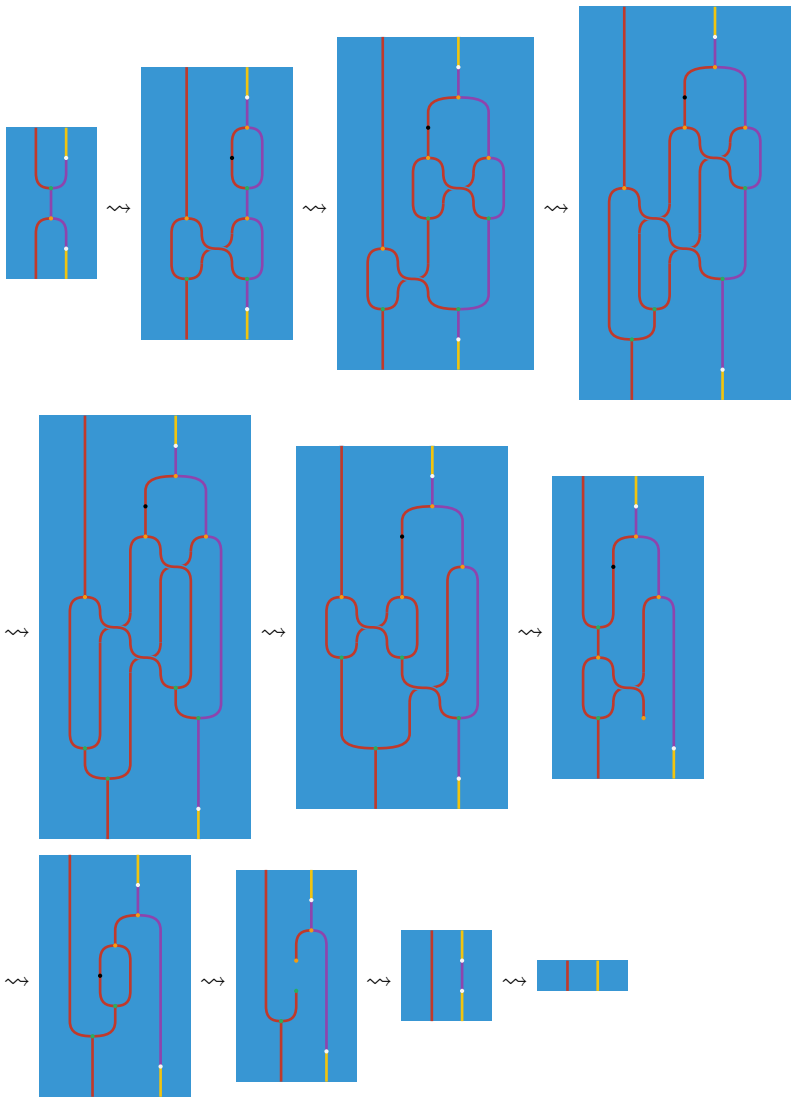


Figure 27 The proof of $\psi \circ \xi = 1_{H \otimes_M H}$ in Theorem 5.

Equational Theories and Validity for Logically Constrained Term Rewriting

Takahito Aoto
Niigata University, Japan

Naoki Nishida
Nagoya University, Japan

Jonas Schöpf
University of Innsbruck, Austria

Abstract

Logically constrained term rewriting is a relatively new formalism where rules are equipped with constraints over some arbitrary theory. Although there are many recent advances with respect to rewriting induction, completion, complexity analysis and confluence analysis for logically constrained term rewriting, these works solely focus on the syntactic side of the formalism lacking detailed investigations on semantics. In this paper, we investigate a semantic side of logically constrained term rewriting. To this end, we first define constrained equations, constrained equational theories and validity of the former based on the latter. After presenting the relationship of validity and conversion of rewriting, we then construct a sound inference system to prove validity of constrained equations in constrained equational theories. Finally, we give an algebraic semantics, which enables one to establish invalidity of constrained equations in constrained equational theories. This algebraic semantics derives a new notion of consistency for constrained equational theories.

2012 ACM Subject Classification Theory of computation → Equational logic and rewriting

Keywords and phrases constrained equation, constrained equational theory, logically constrained term rewriting, algebraic semantics, consistency

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.31

Related Version *Full Version*: `https://arxiv.org/abs/2405.01174` [1]

Funding This research was supported by the FWF (Austrian Science Fund) project I 5943-N and JSPS-FWF Grant Number JPJSBP120222001.
Takahito Aoto: JSPS KAKENHI Grant Numbers 21K11750, 24K14817
Naoki Nishida: JSPS KAKENHI Grant Number 24K02900
Jonas Schöpf: FWF (Austrian Science Fund) project I 5943-N

Acknowledgements We thank the anonymous reviewers for their valuable feedback, which improved the paper.

1 Introduction

Logically constrained term rewriting is a relatively new formalism building upon many-sorted term rewriting and built-in theories. The rules of a logically constrained term rewrite system (LCTRS, for short) are equipped with constraints over some arbitrary theory, which have to be fulfilled in order to apply rules in rewrite steps. This formalism intends to live up with data structures which are often difficult to represent in basic rewriting, such as integers and bit-vectors, with the help of external provers and their built-in theories.

Logical syntax and semantics are often conceived as two sides of the same coin. This is not exceptional, especially for equational logic in which term rewriting lies. On the other hand, although there are many recent advances in rewriting induction [9], completion [20],

complexity analysis [21], confluence analysis [13, 15, 17] and (all-path) reachability [4, 12, 11] for LCTRSs, these works solely focus on the syntactic side of the formalism, lacking detailed investigations on semantics.

In this paper, we investigate a semantic side of the LCTRS formalism. To this end, we first define *constrained equations* (CEs, for short) and *constrained equational theories* (CE-theories, for short). In (first-order) term rewriting, the equational version of rewrite rules is obtained by removing the orientation of the rules. However, in the case of LCTRSs, if we consider a constrained rule $\ell \to r\ [\varphi]$ and relate this naively to a CE $\ell \approx r\ [\varphi]$, which does not distinguish between left- and right-hand sides, we lose information about the restriction on the possible instantiation of variables. This motivates us to add an explicit set X to each CE $\ell \approx r\ [\varphi]$ as $\Pi X.\, \ell \approx r\ [\varphi]$[1] – we name variables in X as *logical variables* with respect to the equation. A CE-theory is then defined as a set of CEs. Similar to the rewrite steps of LCTRSs, we define validity by convertibility if all logical variables are instantiated by values – we denote this notion of validity as *CE-validity* for clarity.

After establishing fundamental properties of the CE-validity, we present its relation to the conversion of rewriting. However, the conversion of rewriting is useful in general to establish the validity of arbitrary CEs. This motivates us to introduce $\mathbf{CEC}_0$, an inference calculus for deriving valid CEs. After demonstrating the usefulness of $\mathbf{CEC}_0$ via some derivations, we present a soundness theorem for the calculus. We also show a partial completeness result, followed by a discussion why our system seems incomplete. Afterwards we consider the opposite question, namely how to prove that a CE is not valid for a particular CE-theory. To this end, we introduce an algebraic semantics that captures CE-validity. We give a natural notion of models for CE-theory, which we call *CE-algebras*. We establish soundness and completeness with respect to CE-validity for this.

Figure 1 presents the relationships between the introduced notions and results of this paper. The following concrete contributions are covered in this paper:

1. We propose a formulation of CEs and CE-theories.
2. On top of that we devise a notion of validity of a CE for a CE-theory $\mathcal{E}$, which we call CE-validity.
3. We give a proof system $\mathbf{CEC}_0$, and show soundness (Theorem 4.6) and a partial completeness result (Theorem 4.10) with respect to CE-validity.
4. We give a notion of CE-algebras and based on it we define algebraic semantics, which is sound (Theorem 5.6) and complete (Theorem 5.17) with respect to CE-validity for *consistent* CE-theories.

We want to discuss some highlights of the last item for readers who are familiar with algebraic semantics of equational logic. First of all, our definition of CE-algebras admits extended underlying models, contrast to those that precisely contain the same underlying models; we will demonstrate why this generalization is required to obtain the completeness result. To reflect this definition, it was necessary to modify the definition of congruence relation to a non-standard one. Also, the notion of consistency with respect to values arises to guarantee this modified notion of congruence in the term algebras. Moreover, it also turns out that value-consistency is equivalent to a more intuitive notion of consistency.

The remainder of the paper is organized as follows. In the next section, we briefly explain the LCTRS formalism, and present some basic lemmas that are necessary for our proofs. Section 3 introduces the notion of CEs, CE-theories and CE-validity, and presents basic

[1] In the literature, some other approaches exist. The computation of critical pairs is also prone of losing information [17]. They solved it by adding dummy constraints $x = x$ to the critical pair. Another approach was proposed in [20] where $\mathcal{LV}\mathsf{ar}(\ell \approx r\ [\varphi])$ was simply defined as $\mathcal{V}\mathsf{ar}(\varphi)$.

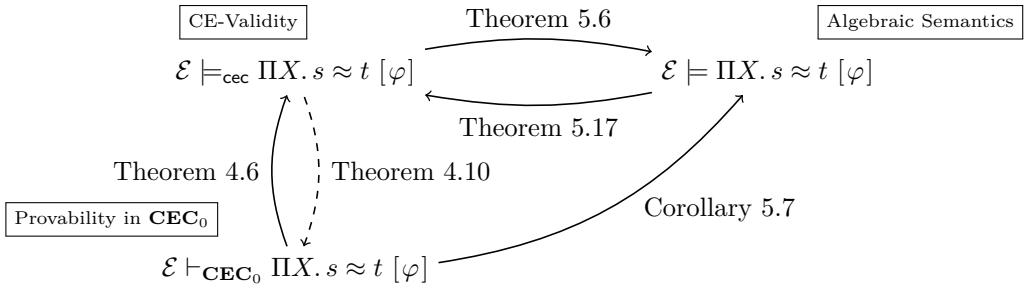


Figure 1 An overview of the main results of this paper.

properties on CE-validity and its relation to the conversion of rewriting. Section 4 is devoted to our inference system $\mathbf{CEC}_0$, including its soundness and partial completeness with respect to CE-validity. In Section 5, we present algebraic semantics, and soundness and completeness results with respect to CE-validity. Before concluding this paper in Section 7, we briefly describe related work in Section 6. We provide only brief proof sketches of selected results in this paper. However, all detailed proofs are given in the full version of this paper [1].

2 Preliminaries

In this section, we briefly recall LCTRSs [13, 9, 17]. Familiarity with the basic notions on mathematical logic [8, 19] and term rewriting [2, 16] is assumed.

The (sorted) signature of an LCTRS is given by the set $\mathcal{S}$ of sorts and the set $\mathcal{F}$ of $\mathcal{S}$-sorted function symbols. Each $f \in \mathcal{F}$ is equipped with a sort declaration $f\colon \tau_1 \times \cdots \times \tau_n \to \tau_0$ with $\tau_0, \ldots, \tau_n \in \mathcal{S}$; $\tau_1 \times \cdots \times \tau_n \to \tau_0$ is said to be the sort of f, and we denote by $\mathcal{F}^{\tau_1 \times \cdots \times \tau_n \to \tau_0}$ the set of function symbols of sort $\tau_1 \times \cdots \times \tau_n \to \tau_0$. For constants of sort $\to \tau$ we drop $\to$ and write τ instead of $\to \tau$. The set of $\mathcal{S}$-sorted variables is denoted by $\mathcal{V}$ and the set of $\mathcal{S}$-sorted terms over $\mathcal{F}, \mathcal{V}$ is $\mathcal{T}(\mathcal{F}, \mathcal{V})$. For each $\tau \in \mathcal{S}$, we denote by $\mathcal{V}^\tau$ the set of variables of sort τ and by $\mathcal{T}(\mathcal{F}, \mathcal{V})^\tau$ the set of terms of sort τ; we also write t^τ for a term t such that $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})^\tau$. The set of variables occurring in a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ is denoted by $\mathcal{V}\mathsf{ar}(t)$ and can be restricted by a set of sorts T with $\mathcal{V}\mathsf{ar}^T(t) = \{x^\tau \in \mathcal{V}\mathsf{ar}(t) \mid \tau \in T\}$. A substitution σ is a mapping $\mathcal{V} \to \mathcal{T}(\mathcal{F}, \mathcal{V})$ such that $\mathcal{D}\mathsf{om}(\sigma) = \{x \in \mathcal{V} \mid x \neq \sigma(x)\}$ is finite and $\sigma(x^\tau) \in \mathcal{T}(\mathcal{F}, \mathcal{V})^\tau$ is satisfied for all $x \in \mathcal{D}\mathsf{om}(\sigma)$.

In the LCTRS formalism, sorts are divided into two categories, that is, each sort $\tau \in \mathcal{S}$ is either a *theory sort* or a *term sort*, where we denote by $\mathcal{S}_{\mathsf{th}}$ the set of theory sorts and by $\mathcal{S}_{\mathsf{te}}$ the set of term sorts, i.e. $\mathcal{S} = \mathcal{S}_{\mathsf{th}} \uplus \mathcal{S}_{\mathsf{te}}$. Accordingly, the set of variables is partitioned as $\mathcal{V} = \mathcal{V}_{\mathsf{th}} \uplus \mathcal{V}_{\mathsf{te}}$ by letting $\mathcal{V}_{\mathsf{th}}$ for the set of variables of sort $\tau \in \mathcal{S}_{\mathsf{th}}$ and $\mathcal{V}_{\mathsf{te}}$ for the set of variables of sort $\tau \in \mathcal{S}_{\mathsf{te}}$. Furthermore, we assume each function symbol $f \in \mathcal{F}$ is either a theory symbol or a term symbol, where all former symbols $f\colon \tau_1 \times \cdots \times \tau_n \to \tau_0$ need to satisfy $\tau_i \in \mathcal{S}_{\mathsf{th}}$ for all $0 \leqslant i \leqslant n$. The sets of theory and term symbols are denoted by $\mathcal{F}_{\mathsf{th}}$ and $\mathcal{F}_{\mathsf{te}}$, respectively: $\mathcal{F} = \mathcal{F}_{\mathsf{th}} \uplus \mathcal{F}_{\mathsf{te}}$. Throughout the paper, we consider signatures consisting of four components $\langle \mathcal{S}_{\mathsf{th}}, \mathcal{S}_{\mathsf{te}}, \mathcal{F}_{\mathsf{th}}, \mathcal{F}_{\mathsf{te}} \rangle$. In some cases term/theory signature stands for the two respective term/theory components of such a signature.

An LCTRS is also equipped with a model over the sorts $\mathcal{S}_{\mathsf{th}}$ and the symbols $\mathcal{F}_{\mathsf{th}}$, which is given by $\mathcal{M} = \langle \mathcal{I}, \mathcal{J} \rangle$, where $\mathcal{I}$ assigns each $\tau \in \mathcal{S}_{\mathsf{th}}$ a *non-empty* set $\mathcal{I}(\tau)$, specifying its domain, and $\mathcal{J}$ assigns each $f\colon \tau_1 \times \cdots \times \tau_n \to \tau_0 \in \mathcal{F}_{\mathsf{th}}$ an interpretation function $\mathcal{J}(f)\colon \mathcal{I}(\tau_1) \times \cdots \times \mathcal{I}(\tau_n) \to \mathcal{I}(\tau_0)$. In particular, $\mathcal{J}(c) \in \mathcal{I}(\tau)$ for any constant $c \in \mathcal{F}_{\mathsf{th}}^\tau$. We suppose for each $\tau \in \mathcal{S}_{\mathsf{th}}$, there exists a subset $\mathcal{V}\mathsf{al}_\tau \subseteq \mathcal{F}_{\mathsf{th}}^\tau$ of constants of sort τ such

that (the restriction of) $\mathcal{J}$ to $\mathcal{V}\mathsf{al}_\tau$ forms a bijection $\mathcal{V}\mathsf{al}_\tau \cong \mathcal{I}(\tau)$. We let $\mathcal{V}\mathsf{al} = \bigcup_{\tau\in\mathcal{S}_{\mathsf{th}}} \mathcal{V}\mathsf{al}_\tau$, whose elements are called *values*. For simplicity, we do not distinguish between $c \in \mathcal{V}\mathsf{al}$ and $\mathcal{J}(c)$. Note that, in [13, 9], an arbitrary overlap between term and theory symbols is allowed provided it is covered by values. For simplicity, we assume $\mathcal{F}_{\mathsf{th}} \cap \mathcal{F}_{\mathsf{te}} = \varnothing$.

A *valuation* over a model $\mathcal{M} = \langle \mathcal{I}, \mathcal{J}\rangle$ is a family $\rho = (\rho_\tau)_{\tau\in\mathcal{S}_{\mathsf{th}}}$ of mappings $\rho_\tau \colon \mathcal{V}^\tau \to \mathcal{I}(\tau)$. The *interpretation* $[\![t]\!]_{\mathcal{M},\rho} \in \mathcal{I}(\tau)$ of a term $t^\tau \in \mathcal{T}(\mathcal{F}_{\mathsf{th}}, \mathcal{V})$ in the model $\mathcal{M}$ with respect to the valuation $\rho = (\rho_\tau)_{\tau\in\mathcal{S}_{\mathsf{th}}}$ is inductively defined as follows: $[\![x^\tau]\!]_{\mathcal{M},\rho} = \rho^\tau(x)$ and $[\![f(t_1,\dots,t_n)]\!]_{\mathcal{M},\rho} = \mathcal{J}(f)([\![t_1]\!]_{\mathcal{M},\rho},\dots,[\![t_n]\!]_{\mathcal{M},\rho})$. We abbreviate $[\![t]\!]_{\mathcal{M},\rho}$ as $[\![t]\!]_\rho$ if $\mathcal{M}$ is known from the context. Furthermore, for any ground term $t \in \mathcal{T}(\mathcal{F}_{\mathsf{th}})$, the valuation ρ has no impact on the interpretation $[\![t]\!]_\rho$ which can be safely ignored and written as $[\![t]\!]$.

We suppose a special sort $\mathsf{Bool} \in \mathcal{S}_{\mathsf{th}}$ such that $\mathcal{I}(\mathsf{Bool}) = \mathbb{B} = \{\mathsf{true}, \mathsf{false}\}$, and usual logical connectives $\neg, \wedge, \vee, \dots \in \mathcal{F}_{\mathsf{th}}$ with their default sorts. We assume that there exists for each $\tau \in \mathcal{S}_{\mathsf{th}}$ an equality symbol $=_\tau$ of sort $\tau \times \tau \to \mathsf{Bool}$ in $\mathcal{F}_{\mathsf{th}}$. For brevity we will omit $_\tau$ from $=_\tau$. We assume, for all of these theory symbols, that their interpretation functions model their default semantics. The terms in $\mathcal{T}(\mathcal{F}_{\mathsf{th}}, \mathcal{V})^{\mathsf{Bool}}$ are called *logical constraints*.[2] Note that $\mathcal{V}\mathsf{ar}(\varphi) \subseteq \mathcal{V}_{\mathsf{th}}$ for any logical constraint φ, thus in this case $\mathcal{T}(\mathcal{F}_{\mathsf{th}}, \mathcal{V})^{\mathsf{Bool}} = \mathcal{T}(\mathcal{F}_{\mathsf{th}}, \mathcal{V}_{\mathsf{th}})^{\mathsf{Bool}}$. We say that a logical constraint φ is over a set $X \subseteq \mathcal{V}_{\mathsf{th}}$ of theory variables if $\mathcal{V}(\varphi) \subseteq X$. A logical constraint φ is said to be *valid* in a model $\mathcal{M}$, written as $\models_{\mathcal{M}} \varphi$ (or $\models \varphi$ when the model $\mathcal{M}$ is known from the context), if $[\![\varphi]\!]_{\mathcal{M},\rho} = \mathsf{true}$ for any valuation ρ over the model $\mathcal{M}$. Considering the bijection $\mathcal{V}\mathsf{al}_\tau \cong \mathcal{I}(\tau)$, an arbitrary substitution σ is equivalent to a valuation ρ. Suppose that $\mathcal{VD}\mathsf{om}(\sigma) = \{x \in \mathcal{D}\mathsf{om}(\sigma) \mid \sigma(x) \in \mathcal{V}\mathsf{al}\}$ and $\mathcal{V}\mathsf{ar}(\varphi) \subseteq \mathcal{VD}\mathsf{om}(\sigma)$. Then the substitution σ can be seen as a valuation over φ, and $\models_{\mathcal{M}} \varphi\sigma$ coincides with $[\![\varphi]\!]_{\mathcal{M},\sigma} = \mathsf{true}$. More generally, we have the following.

▶ **Lemma 2.1.** *Let $t \in \mathcal{T}(\mathcal{F}_{\mathsf{th}}, \mathcal{V}_{\mathsf{th}})$, ρ a valuation, and σ a substitution.*

1. *Suppose $\sigma(x) \in \mathcal{T}(\mathcal{F}_{\mathsf{th}}, \mathcal{V}_{\mathsf{th}})$ for all $x \in \mathcal{V}_{\mathsf{th}}$. Let $[\![\sigma]\!]_{\mathcal{M},\rho}$ be a valuation defined as $[\![\sigma]\!]_{\mathcal{M},\rho}(x) = [\![\sigma(x)]\!]_{\mathcal{M},\rho}$. Then, $[\![t]\!]_{\mathcal{M},[\![\sigma]\!]_{\mathcal{M},\rho}} = [\![t\sigma]\!]_{\mathcal{M},\rho}$.*
2. *Suppose that $\mathcal{V}\mathsf{ar}(t) \subseteq \mathcal{VD}\mathsf{om}(\sigma)$. Then, $[\![t]\!]_{\mathcal{M},\hat\sigma} = [\![t\sigma]\!]_{\mathcal{M}}$, where the valuation $\hat\sigma$ is defined by $\hat\sigma(x^\tau) = \xi(\sigma(x)) \in \mathcal{I}(\tau)$ for $x \in \mathcal{VD}\mathsf{om}(\sigma)$, where ξ is a bijection $\mathcal{V}\mathsf{al}^\tau \cong \mathcal{I}(\tau)$.*

Proof (Sketch).

1. Use structural induction on $t \in \mathcal{T}(\mathcal{F}_{\mathsf{th}}, \mathcal{V}_{\mathsf{th}})$.
2. Similar to **1**, using the assumption $\mathcal{V}\mathsf{ar}(t) \subseteq \mathcal{VD}\mathsf{om}(\sigma)$. ◀

From Lemma 2.1 the following characterizations, which are used later on, are obtained. Note that $\models \varphi = \mathsf{true}$ ($\models \varphi = \mathsf{false}$) if and only if $\models \varphi$ ($\models \neg\varphi$), for a logical constraint φ.

▶ **Lemma 2.2.** *Let φ be a logical constraint.*

1. *$\models_{\mathcal{M}} \varphi$ if and only if $\models_{\mathcal{M}} \varphi\sigma$ for all substitutions σ such that $\mathcal{V}\mathsf{ar}(\varphi) \subseteq \mathcal{VD}\mathsf{om}(\sigma)$.*
2. *If $\models_{\mathcal{M}} \varphi$, then $\models_{\mathcal{M}} \varphi\sigma$ for all substitutions σ such that $\sigma(x) \in \mathcal{T}(\mathcal{F}_{\mathsf{th}}, \mathcal{V}_{\mathsf{th}})$ for all $x \in \mathcal{V}\mathsf{ar}(\varphi) \cap \mathcal{D}\mathsf{om}(\sigma)$.*
3. *The following statements are equivalent: (1) $\models_{\mathcal{M}} \neg\varphi$, (2) $\not\models_{\mathcal{M}} \varphi\sigma$ for all substitutions σ such that $\mathcal{V}\mathsf{ar}(\varphi) \subseteq \mathcal{VD}\mathsf{om}(\sigma)$, and (3) $\sigma \models_{\mathcal{M}} \varphi$ for no substitution σ.*

Here, $\sigma \models_{\mathcal{M}} \varphi$ denotes that $\mathcal{V}\mathsf{ar}(\varphi) \subseteq \mathcal{VD}\mathsf{om}(\sigma)$ and $\models_{\mathcal{M}} \varphi\sigma$ hold.

---

[2] Logical constraints are quantifier-free, which is not restrictive: Consider, for example, a formula $\forall x.\ \varphi$ with n free variables $x_1,\dots,x_n$ and a quantifier-free formula φ. By introducing an n-ary predicate symbol p defined as $[\![p(x_1,\dots,x_n)]\!]_{\mathcal{M},\rho} = [\![\forall x.\ \varphi]\!]_{\mathcal{M},\rho}$, we can replace the formula by the quantifier-free formula $p(x_1,\dots,x_n)$. Clearly, this applies to arbitrary first-order formulas. Another approach can be seen in [9, Section 2.2].

Proof (Sketch).

1. ($\Rightarrow$) Let σ be a substitution such that $\mathcal{V}\mathsf{ar}(\varphi) \subseteq \mathcal{VD}\mathsf{om}(\sigma)$, and $\hat{\sigma}$ be defined as in Lemma 2.1. Then, $[\![\varphi]\!]_{\mathcal{M},\hat{\sigma}} = \mathsf{true}$, and hence $[\![\varphi\sigma]\!]_{\mathcal{M}} = \mathsf{true}$ by Lemma 2.1. Therefore, $\models_{\mathcal{M}} \varphi\sigma$. ($\Leftarrow$) Let ρ be a valuation over a model $\mathcal{M} = \langle \mathcal{I}, \mathcal{J} \rangle$. Then, in the view of $\mathcal{V}\mathsf{al}_\tau \cong \mathcal{I}(\tau)$, we can take a substitution $\check{\rho}$ given by $\check{\rho}(x) = \rho(x) \in \mathcal{V}\mathsf{al}$ for all $x \in \mathcal{V}\mathsf{ar}(\varphi)$. Then, use Lemma 2.1 to obtain $[\![\varphi]\!]_{\mathcal{M},\rho} = [\![\varphi]\!]_{\mathcal{M},\hat{\check{\rho}}} = \mathsf{true}$, from which $\models_{\mathcal{M}} \varphi$ follows.
2. Take a substitution σ' such that $\sigma'(x) = \sigma(x)$ for $x \in \mathcal{V}\mathsf{ar}(\varphi)$ and $\sigma'(x) = x$ otherwise. Then, using Lemma 2.1, we have $[\![\varphi]\!]_{[\![\sigma]\!]_\rho} = [\![\varphi]\!]_{[\![\sigma']\!]_\rho} = [\![\varphi\sigma']\!]_\rho = [\![\varphi\sigma]\!]_\rho$. Thus, $[\![\varphi\sigma]\!]_\rho = \mathsf{true}$ for any ρ. Therefore, $\models_{\mathcal{M}} \varphi\sigma$.
3. Use **1**. ◀

LCTRSs admit special rewrite steps over $\mathcal{T}(\mathcal{F}, \mathcal{V})$ specified by the underlying model $\mathcal{M} = \langle \mathcal{I}, \mathcal{J} \rangle$. Such rewrite steps are called *calculation steps* and denoted by $s \to_{\mathsf{calc}} t$, which is defined as follows: $s \to_{\mathsf{calc}} t$ if $s = C[f(c_1, \ldots, c_n)]$ and $t = C[c_0]$ for $f \in \mathcal{F}_{\mathsf{th}} \setminus \mathcal{V}\mathsf{al}$ and $c_0, \ldots, c_n \in \mathcal{V}\mathsf{al}$ with $c_0 = \mathcal{J}(f)(c_1, \ldots, c_n)$ and a context C. The following lemma connects calculation steps and interpretations over ground theory terms $\mathcal{T}(\mathcal{F}_{\mathsf{th}})$. In the following $s \to^! t$ is used for $s \to^* t$ with t being a normal form with respect to $\to$.

▶ **Lemma 2.3.** *Let $s, t \in \mathcal{T}(\mathcal{F}_{\mathsf{th}})$. Then, all of the following holds:*

1. $[\![t]\!] \in \mathcal{V}\mathsf{al}$,
2. $t \to^!_{\mathsf{calc}} [\![t]\!]$,
3. $s \to^*_{\mathsf{calc}} t$ *implies* $[\![s]\!] = [\![t]\!]$, *and*
4. $s \leftrightarrow^*_{\mathsf{calc}} t$ *if and only if* $[\![s]\!] = [\![t]\!]$.

Proof (Sketch).

1. This claim follows as $[\![t^\tau]\!] \in \mathcal{I}(\tau) \cong \mathcal{V}\mathsf{al}^\tau$.
2. Show $t \to^*_{\mathsf{calc}} [\![t]\!]$ by structural induction on t. Then, the claim follows, since values are normal forms with respect to calculation steps.
3. We use the fact that the set of calculation rules forms a confluent LCTRS [13]. Since $s \to^!_{\mathsf{calc}} [\![s]\!]$ and $t \to^!_{\mathsf{calc}} [\![t]\!]$ from **2**, $s \to^*_{\mathsf{calc}} t$ implies $[\![s]\!] = [\![t]\!]$ by confluence.
4. The *only-if* part follows from **3**, and the *if* part follows from **1**. ◀

The other type of rewrite steps in LCTRSs are rule steps specified by rewrite rules. Let us fix a signature $\langle \mathcal{S}_{\mathsf{th}}, \mathcal{S}_{\mathsf{te}}, \mathcal{F}_{\mathsf{th}}, \mathcal{F}_{\mathsf{te}} \rangle$. A constrained rule of an LCTRS is a triple $\ell \to r\ [\varphi]$ of terms ℓ, r with the same sort satisfying $root(\ell) \in \mathcal{F}_{\mathsf{te}}$ and a logical constraint φ. We define $\mathcal{LV}\mathsf{ar}(\ell \to r\ [\varphi]) = (\mathcal{V}\mathsf{ar}(r) \setminus \mathcal{V}\mathsf{ar}(\ell)) \cup \mathcal{V}\mathsf{ar}(\varphi)$, whose members are called *logical variables* of the rule. The intention is that the logical variables of rules in LCTRSs are required to be instantiated only by values. Let us also fix a model $\mathcal{M}$. Then, a substitution γ is said to *respect a rewrite rule* $\ell \to r\ [\varphi]$ if $\mathcal{LV}\mathsf{ar}(\ell \to r\ [\varphi]) \subseteq \mathcal{VD}\mathsf{om}(\gamma)$ and $\models_{\mathcal{M}} \varphi\gamma$. Using this notation, a rule step $s \to_{\mathsf{rule}} t$ over the model $\mathcal{M}$ by the rewrite rule $\ell \to r\ [\varphi]$ is given as follows: $s \to_{\mathsf{rule}} t$ if and only if $s = C[\ell\gamma]$ and $t = C[r\gamma]$ for some context C and some substitution γ that respects the rewrite rule $\ell \to r\ [\varphi]$.

Finally, a *logically constrained term rewrite system* (LCTRS, for short) consists of a signature $\Sigma = \langle \mathcal{S}_{\mathsf{th}}, \mathcal{S}_{\mathsf{te}}, \mathcal{F}_{\mathsf{th}}, \mathcal{F}_{\mathsf{te}} \rangle$, a model $\mathcal{M}$ over $\Sigma_{\mathsf{th}} = \langle \mathcal{S}_{\mathsf{th}}, \mathcal{F}_{\mathsf{th}} \rangle$ (which induces the set $\mathcal{V}\mathsf{al} \subseteq \mathcal{F}_{\mathsf{th}}$ of values) and a set $\mathcal{R}$ of constrained rules over the signature Σ. All this together defines rewrite steps consisting of calculation steps and rule steps. In a practical setting, often some predefined (semi-)decidable theories are assumed and used as model $\mathcal{M}$ and theory signature $\langle \mathcal{S}_{\mathsf{th}}, \mathcal{F}_{\mathsf{th}} \rangle$. An example of such a theory is *linear integer arithmetic*, whose model consists of standard boolean functions and the set of integers including standard predefined functions on them. From this point of view, we call the triple $\mathfrak{U} = \langle \mathcal{S}_{\mathsf{th}}, \mathcal{F}_{\mathsf{th}}, \mathcal{M} \rangle$

of the theory signature and its respective model the *underlying model* or *background theory* of the LCTRS. We also denote an LCTRS as $\langle \mathcal{M}, \mathcal{R} \rangle$ with an implicit signature or $\langle \mathcal{M}, \mathcal{R} \rangle$ over the signature $\Sigma = \langle \mathcal{S}_{\mathsf{th}}, \mathcal{S}_{\mathsf{te}}, \mathcal{F}_{\mathsf{th}}, \mathcal{F}_{\mathsf{te}} \rangle$ for an explicit signature.

3 Validity of Constrained Equational Theories

In this section, we introduce validity of constrained equational theories (CE-validity), which is a key concept used throughout the paper. Subsequently, we present fundamental properties of CE-validity, and show their relation to the conversion of rewriting.

3.1 Constrained Equational Theory and Its Validity

In this subsection, after introducing the notion of CEs, we define equational systems, which are sets of CEs, and rewriting with respect to such systems. This gives an equational version of the rewrite step in LCTRSs. Furthermore, based on these notions, we define the validity of CEs.

Recall that logical variables of a constrained rule are those which are only allowed to be instantiated by values. As we have seen in the previous section, rewrite steps of LCTRSs depend on the correct instantiation of the logical variables of the applied rule. However, the sets of logical variables $\mathcal{LV}\mathsf{ar}(\ell \to r\ [\varphi])$ and $\mathcal{LV}\mathsf{ar}(r \to \ell\ [\varphi])$ are not necessarily equivalent, and the CE $\ell \approx r\ [\varphi]$ alone does not suffice to specify the correct logical variables. This motivates us to add an explicit set X to the CE $\ell \approx r\ [\varphi]$ as $\Pi X.\, \ell \approx r\ [\varphi]$ which specifies its logical variables.

▶ **Definition 3.1** (constrained equation). *Let $\Sigma_{\mathsf{te}} = \langle \mathcal{S}_{\mathsf{te}}, \mathcal{F}_{\mathsf{te}} \rangle$ be a term signature over the underlying model $\mathfrak{U} = \langle \mathcal{S}_{\mathsf{th}}, \mathcal{F}_{\mathsf{th}}, \mathcal{M} \rangle$. A* constrained equation *(CE, for short) over $\mathfrak{U}$ and Σ_{te} is a quadruple $\Pi X.\, s \approx t\ [\varphi]$ where s, t are terms with the same sort, φ is a logical constraint, and $X \subseteq \mathcal{V}_{\mathsf{th}}$ is a set of theory variables satisfying $\mathcal{V}\mathsf{ar}(\varphi) \subseteq X$. A* logically constrained equational system *(LCES, for short) is a set of CEs. We abbreviate $\Pi X.\, s \approx t\ [\varphi]$ to $s \approx t\ [\varphi]$ if $\mathcal{V}\mathsf{ar}(\varphi) = X$. A CE $\Pi X.\, s \approx t\ [\mathsf{true}]$ is abbreviated to $\Pi X.\, s \approx t$.*

We remark that a constrained rewrite rule $\ell \to r\ [\varphi]$ is naturally encoded as a CE $\Pi X.\, \ell \approx r\ [\varphi]$ by taking $X = \mathcal{LV}\mathsf{ar}(\ell \to r\ [\varphi])$. Furthermore, let us illustrate the aforementioned issues, without an explicit set of logical variables, by an example.

▶ **Example 3.2.** Consider the LCTRS $\mathcal{R}$ over the theory of integer arithmetic and its (labeled) rules

$$\alpha\colon \mathsf{f}(x, y) \to \mathsf{g}(z)\ [x = 1] \qquad\qquad \beta\colon \mathsf{g}(z) \to \mathsf{f}(x, y)\ [x = 1]$$

with their sets of logical variables $\mathcal{LV}\mathsf{ar}(\alpha) = \{x, z\}$ and $\mathcal{LV}\mathsf{ar}(\beta) = \{x, y\}$. Transforming them naively into the CE $\mathsf{f}(x, y) \approx \mathsf{g}(z)\ [x = 1]$ and $\mathsf{g}(z) \approx \mathsf{f}(x, y)\ [x = 1]$ would give the set of logical variables $\{x\}$ for both. We use the notion of logical variables in Winkler and Middeldorp [20], where the set of logical variables of a CE consists of the variables appearing in the constraint. Obviously, we lose concrete information about logical variables of the original rules. Clearly, in our notion this information remains intact: $\Pi\{x, z\}.\, \mathsf{f}(x, y) \approx \mathsf{g}(z)\ [x = 1]$ and $\Pi\{x, y\}.\, \mathsf{g}(z) \approx \mathsf{f}(x, y)\ [x = 1]$. Note that variables appearing solely in the set of logical variables and not in the CE have no effect but are allowed. For example, in the CE $\Pi\{x, z, z'\}.\, \mathsf{f}(x, y) \approx \mathsf{g}(z)\ [x = 1]$ the logical variable z' has no effect and could be dropped.

In the following we extend the notion of rewrite steps by using CEs instead of rewrite rules.

▶ **Definition 3.3** ($\leftrightarrow_{\mathcal{E}}$). *Let $\mathcal{E}$ be an LCES over the underlying model $\mathfrak{U} = \langle \mathcal{S}_{\mathsf{th}}, \mathcal{F}_{\mathsf{th}}, \mathcal{M}\rangle$ and the term signature $\Sigma_{\mathsf{te}} = \langle \mathcal{S}_{\mathsf{te}}, \mathcal{F}_{\mathsf{te}}\rangle$. For terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, we define a* rule *step $s \leftrightarrow_{\mathsf{rule},\mathcal{E}} t$ if $s = C[\ell\sigma]$ and $t = C[r\sigma]$ (or vice versa) for some CE $\Pi X.\, \ell \approx r\ [\varphi] \in \mathcal{E}$ and some X-valued substitution σ such that $\models_{\mathcal{M}} \varphi\sigma$. Here, a substitution is said to be* X-valued *if $X \subseteq \mathcal{VD}\mathsf{om}(\sigma)$. We let $\leftrightarrow_{\mathcal{E}} = \leftrightarrow_{\mathsf{calc}} \cup \leftrightarrow_{\mathsf{rule},\mathcal{E}}$, where $\leftrightarrow_{\mathsf{calc}}$ is the symmetric closure of the calculation steps $\rightarrow_{\mathsf{calc}}$ specified by $\mathcal{M}$.*

We give examples on rewriting with CEs.

▶ **Example 3.4.** Consider integer arithmetic as underlying model $\mathcal{M}$. We consider the term sorts $\mathcal{S}_{\mathsf{te}} = \{\mathsf{Unit}\}$ and the term signature $\mathcal{F}_{\mathsf{te}} = \{\mathsf{cong}\colon \mathsf{Int} \to \mathsf{Unit}\}$ where Int is the respective sort of the integers. The set $\mathcal{E}$ of CEs consists of $\{\mathsf{cong}(x) \approx \mathsf{cong}(y)\ [\mathsf{mod}(x, 12) = \mathsf{mod}(y, 12)]\}$. Arithmetic values in intermediate steps of rewrite sequences wrapped in cong have the property that they are *congruent modulo 12* and thus $\mathcal{E}$ simulates modular arithmetic with modulus 12. Consider the following sequence:

$$\mathsf{cong}(7 + 31) \leftrightarrow_{\mathsf{calc}} \mathsf{cong}(38) \leftrightarrow_{\mathsf{rule},\mathcal{E}} \mathsf{cong}(14)$$

From this we conclude that $7 + 31$, which gives 38, and 14 are congruent modulo 12. Note that the rule step $\leftrightarrow_{\mathsf{rule},\mathcal{E}}$ does not allow to directly convert $\mathsf{cong}(7 + 31)$ and $\mathsf{cong}(14)$.

▶ **Example 3.5.** Consider integer arithmetic as the underlying model $\mathcal{M}$. We take a term signature $\mathcal{S}_{\mathsf{te}} = \{\mathsf{G}\}$ and $\mathcal{F}_{\mathsf{te}} = \{\mathsf{e}\colon \mathsf{G}, \mathsf{inv}\colon \mathsf{G} \to \mathsf{G}, *\colon \mathsf{G} \times \mathsf{G} \to \mathsf{G}, \mathsf{exp}\colon \mathsf{G} \times \mathsf{Int} \to \mathsf{G}\}$. Let the set $\mathcal{E}$ of CEs consist of:

$$\begin{array}{rcl@{\qquad}rcl} (x * y) * z & \approx & x * (y * z) & \mathsf{e} * x & \approx & x \\ \mathsf{inv}(x) * x & \approx & \mathsf{e} & \mathsf{exp}(x, 0) & \approx & \mathsf{e} \\ \mathsf{exp}(x, 1) & \approx & x & \Pi\{n, m\}.\, \mathsf{exp}(x, n) * \mathsf{exp}(x, m) & \approx & \mathsf{exp}(x, m + n) \end{array}$$

As in first-order equational reasoning, one can show $x * \mathsf{e} \overset{*}{\leftrightarrow}_{\mathcal{E}} x$. Thus, $\mathsf{exp}(x, -1) \leftrightarrow_{\mathcal{E}} \mathsf{e} * \mathsf{exp}(x, -1) \leftrightarrow_{\mathcal{E}} (\mathsf{inv}(x) * x) * \mathsf{exp}(x, -1) \leftrightarrow_{\mathcal{E}} \mathsf{inv}(x) * (x * \mathsf{exp}(x, -1)) \leftrightarrow_{\mathcal{E}} \mathsf{inv}(x) * (\mathsf{exp}(x, 1) * \mathsf{exp}(x, -1)) \leftrightarrow_{\mathcal{E}} \mathsf{inv}(x) * \mathsf{exp}(x, 1 + (-1)) \leftrightarrow_{\mathcal{E}} \mathsf{inv}(x) * \mathsf{exp}(x, 0) \leftrightarrow_{\mathcal{E}} \mathsf{inv}(x) * \mathsf{e} \overset{*}{\leftrightarrow}_{\mathcal{E}} \mathsf{inv}(x)$ as expected. This encodes a system of groups with an explicit exponentiation operator exp.

▶ **Example 3.6.** Consider integer arithmetic as the underlying model $\mathcal{M}$. We take a term signature $\mathcal{S}_{\mathsf{te}} = \{\mathsf{Elem}, \mathsf{List}, \mathsf{ElemOp}\}$ and $\mathcal{F}_{\mathsf{te}} = \{\mathsf{nil}\colon \mathsf{List}, \mathsf{cons}\colon \mathsf{Elem} \times \mathsf{List} \to \mathsf{List}, \mathsf{none}\colon \mathsf{ElemOp}, \mathsf{some}\colon \mathsf{Elem} \to \mathsf{ElemOp}, \mathsf{length}\colon \mathsf{List} \to \mathsf{Int}, \mathsf{nth}\colon \mathsf{List} \times \mathsf{Int} \to \mathsf{ElemOp}\}$. Let the set $\mathcal{E}$ of CEs consist of

$$\begin{array}{rcl@{\qquad}rcl@{\qquad}l} \mathsf{length}(\mathsf{nil}) & \approx & 0 & \mathsf{length}(\mathsf{cons}(x, xs)) & \approx & \mathsf{length}(xs) + 1 & \\ \Pi\{n\}.\, \mathsf{nth}(\mathsf{nil}, n) & \approx & \mathsf{none} & \mathsf{nth}(xs, n) & \approx & \mathsf{none} & [n < 0] \\ \mathsf{nth}(\mathsf{cons}(x, xs), 0) & \approx & \mathsf{some}(x) & \mathsf{nth}(\mathsf{cons}(x, xs), n) & \approx & \mathsf{nth}(xs, n - 1) & [n > 0] \end{array}$$

This LCES encodes common list functions that use integers. For program verification purposes, one may deal with the validity problem of a formula such as $\mathsf{nth}(xs, n) \not\approx \mathsf{none} \Leftrightarrow 0 \leqslant n \wedge n < \mathsf{length}(xs)$.

We continue by giving some immediate facts which are used later on.

▶ **Lemma 3.7.** *Let $\mathcal{E}$ be an LCES over the underlying model $\mathfrak{U} = \langle \mathcal{S}_{\mathsf{th}}, \mathcal{F}_{\mathsf{th}}, \mathcal{M}\rangle$ and the term signature $\Sigma_{\mathsf{te}} = \langle \mathcal{S}_{\mathsf{te}}, \mathcal{F}_{\mathsf{te}}\rangle$. Then, all of the following hold:*

1. *$\leftrightarrow_{\mathcal{E}}$ is symmetric,*
2. *$\leftrightarrow_{\mathcal{E}}$ is closed under contexts i.e. $s \leftrightarrow_{\mathcal{E}} t$ implies $C[s] \leftrightarrow_{\mathcal{E}} C[t]$ for any context C, and*
3. *$\leftrightarrow_{\mathcal{E}}$ is closed under substitutions, i.e. $s \leftrightarrow_{\mathcal{E}} t$ implies $s\sigma \leftrightarrow_{\mathcal{E}} t\sigma$ for any substitution σ.*

Proof (Sketch). 1 and 2 are trivial. For 3 the case $s \leftrightarrow_{\mathsf{calc}} t$ is clear. Suppose $s \leftrightarrow_{\mathsf{rule},\mathcal{E}} t$. Then $s = C[\ell\rho]$ and $t = C[r\rho]$ (or vice versa) for some CE $\Pi X.\, \ell \approx r\ [\varphi] \in \mathcal{E}$ and an X-valued substitution ρ such that $\models_{\mathcal{M}} \varphi\rho$. Then $\varphi\rho = (\varphi\rho)\sigma = \varphi(\sigma \circ \rho)$ and hence $\models_{\mathcal{M}} \varphi(\sigma \circ \rho)$. Then, the claim follows, as $s\sigma = C[\ell\rho]\sigma = C\sigma[\ell(\sigma \circ \rho)]$ and $t\sigma = C[r\rho]\sigma = C\sigma[r(\sigma \circ \rho)]$. ◀

We proceed by defining constrained equational theories (CE-theories) and validity of CEs (CE-validity) with respect to a CE-theory.

▶ **Definition 3.8** (constrained equational theory). *A* constrained equational theory *is specified by a triple $\mathfrak{T} = \langle \mathfrak{U}, \Sigma_{\mathsf{te}}, \mathcal{E}\rangle$, where $\mathfrak{U} = \langle \mathcal{S}_{\mathsf{th}}, \mathcal{F}_{\mathsf{th}}, \mathcal{M}\rangle$ is an underlying model, Σ_{te} is a term signature over $\mathfrak{U}$ (as given in the LCTRS formalism), and $\mathcal{E}$ is an LCES over $\mathfrak{U}, \Sigma_{\mathsf{te}}$. If no confusion arises, we refer to the CE-theory by $\langle \mathcal{M}, \mathcal{E}\rangle$, without stating its signature explicitly. We also say that a CE-theory $\langle \mathcal{M}, \mathcal{E}\rangle$ is defined over the signature $\Sigma = \langle \mathcal{S}_{\mathsf{th}}, \mathcal{S}_{\mathsf{te}}, \mathcal{F}_{\mathsf{th}}, \mathcal{F}_{\mathsf{te}}\rangle$ in order to make the signature explicit.*

▶ **Definition 3.9** (CE-validity). *Let $\mathfrak{T} = \langle \mathcal{M}, \mathcal{E}\rangle$ be a CE-theory. Then a CE $\Pi X.\, s \approx t\ [\varphi]$ is said to be a* constrained equational consequence *(CE-consequence, for short)* of $\mathfrak{T}$ *or* valid *(CE-valid, for clarity), written as $\mathfrak{T} \models_{\mathsf{cec}} \Pi X.\, s \approx t\ [\varphi]$, if $s\sigma \overset{*}{\leftrightarrow}_{\mathcal{E}} t\sigma$ for all X-valued substitutions σ such that $\models_{\mathcal{M}} \varphi\sigma$. We write $\mathcal{E} \models_{\mathsf{cec}} \Pi X.\, s \approx t\ [\varphi]$ if $\mathcal{M}$ is known from the context.*

We conclude this subsection with an example on CE-validity.

▶ **Example 3.10.** Consider integer arithmetic as the underlying model $\mathcal{M}$. We take the term signature $\mathcal{F}_{\mathsf{te}} = \{\mathsf{abs}\colon \mathsf{Int} \to \mathsf{Int}, \mathsf{max}\colon \mathsf{Int} \times \mathsf{Int} \to \mathsf{Int}\}$ the set of CEs $\mathcal{E}$ consisting of

$$\begin{array}{rcll} \mathsf{abs}(x) & \approx & -x & [x < 0] \\ \mathsf{max}(x,y) & \approx & x & [x \geqslant y] \end{array} \qquad \begin{array}{rcll} \mathsf{abs}(x) & \approx & x & [x \geqslant 0] \\ \mathsf{max}(x,y) & \approx & y & [x < y] \end{array}$$

The following are valid CE-consequences:

$$\mathfrak{T} \models_{\mathsf{cec}} \Pi\{x\}.\, \mathsf{abs}(x) \approx \mathsf{abs}(-x) \qquad \mathfrak{T} \models_{\mathsf{cec}} \Pi\{x,y\}.\, \mathsf{max}(x,y) \approx \mathsf{max}(y,x)$$
$$\mathfrak{T} \models_{\mathsf{cec}} \Pi\{x,y\}.\, \mathsf{abs}(\mathsf{max}(x,y)) \approx \mathsf{max}(\mathsf{abs}(x), \mathsf{abs}(y))\ [0 \leqslant x \wedge 0 \leqslant y]$$

On the other hand, the CE $\Pi\varnothing.\, \mathsf{abs}(x) \approx \mathsf{abs}(-x)$ is not a valid CE-consequence: For the $\varnothing$-valued identity substitution σ, we have that $\mathsf{abs}(x)\sigma = \mathsf{abs}(x) \not\overset{*}{\leftrightarrow}_{\mathcal{E}} \mathsf{abs}(-x) = \mathsf{abs}(-x)\sigma$.

3.2 Properties of CE-Validity

This subsection covers important properties related to CE-validity, for example, we show that validity forms an equivalence and a congruence relation. Furthermore, we cover in which way it is closed under substitutions and contexts, and how equality can be induced from constraints.

Our first two lemmas follow immediately from the definition of the CE-validity.

▶ **Lemma 3.11.** *Let $\mathfrak{T} = \langle \mathcal{M}, \mathcal{E}\rangle$ be a CE-theory. Then for any $\Pi X.\, s \approx t\ [\varphi] \in \mathcal{E}$, we have $\mathfrak{T} \models_{\mathsf{cec}} \Pi X.\, s \approx t\ [\varphi]$.*

▶ **Lemma 3.12** (congruence). *Let $\mathfrak{T} = \langle \mathcal{M}, \mathcal{E}\rangle$ be a CE-theory. For any set $X \subseteq \mathcal{V}_{\mathsf{th}}$ and logical constraint φ such that $\mathcal{V}\mathsf{ar}(\varphi) \subseteq X$, the binary relation $\mathfrak{T} \models_{\mathsf{cec}} \Pi X.\, \cdot \approx \cdot\ [\varphi]$ over terms is a congruence relation over Σ.*

For stability under substitutions, we differentiate two kinds; for each CE $\Pi X.\, s \approx t\ [\varphi]$, the first one considers substitutions instantiating variables in X; the second one considers substitutions instantiating variables not in X.

▶ **Lemma 3.13** (stability of theory terms). *Let $\mathfrak{T} = \langle \mathcal{M}, \mathcal{E} \rangle$ be a CE-theory. Let $X, Y \subseteq \mathcal{V}_{\mathsf{th}}$ be sets of theory variables and σ a substitution such that $\sigma(y) \in \mathcal{T}(\mathcal{F}_{\mathsf{th}}, X)$ for any $y \in Y$. If $\mathfrak{T} \models_{\mathsf{cec}} \Pi Y.\, s \approx t\ [\varphi]$, then $\mathfrak{T} \models_{\mathsf{cec}} \Pi X.\, s\sigma \approx t\sigma\ [\varphi\sigma]$.*

Proof (Sketch). Take any X-valued substitution θ with $\models_{\mathcal{M}} (\varphi\sigma)\theta$. This gives a Y-valued substitution ξ by defining $\xi(y) = [\![(\theta \circ \sigma)(y)]\!]$ for each $y \in Y$. From Lemma 2.3, we know $(\theta \circ \sigma)(y) \overset{*}{\leftrightarrow}_{\mathcal{E}} \xi(y)$ for any $y \in Y$. We also have $\models_{\mathcal{M}} \varphi\xi$ by Lemma 2.1, and hence $s\xi \overset{*}{\leftrightarrow}_{\mathcal{E}} t\xi$ by assumption. Thus, using Lemma 3.7, we obtain $(s\sigma)\theta = s(\theta \circ \sigma) \overset{*}{\leftrightarrow}_{\mathcal{E}} s\xi \overset{*}{\leftrightarrow}_{\mathcal{E}} t\xi \overset{*}{\leftrightarrow}_{\mathcal{E}} t(\theta \circ \sigma) = (t\sigma)\theta$. ◀

▶ **Lemma 3.14** (general stability). *Let $\langle \mathcal{M}, \mathcal{E} \rangle$ be a CE-theory and σ a substitution such that $\mathcal{D}\mathsf{om}(\sigma) \cap X = \varnothing$. Then, if $\mathcal{E} \models_{\mathsf{cec}} \Pi X.\, s \approx t\ [\varphi]$ then $\mathcal{E} \models_{\mathsf{cec}} \Pi X.\, s\sigma \approx t\sigma\ [\varphi]$.*

Proof (Sketch). Take any X-valued substitution δ such that $\models_{\mathcal{M}} \varphi\delta$. Take $\gamma = \delta \circ \sigma$. From $\mathcal{D}\mathsf{om}(\sigma) \cap X = \varnothing$, we have $\varphi\gamma = \varphi\delta$, and therefore, $s\sigma\delta \overset{*}{\leftrightarrow}_{\mathcal{E}} t\sigma\delta$ holds. ◀

One may expect that $\mathcal{E} \models_{\mathsf{cec}} \Pi X.\, s \approx t\ [\varphi]$ holds for equivalent terms s, t such that $\varphi \Rightarrow s = t$ is valid. In fact, a more general result can be obtained.

▶ **Lemma 3.15** (model consequence). *Let $\langle \mathcal{M}, \mathcal{E} \rangle$ be a CE-theory, $X \subseteq \mathcal{V}_{\mathsf{th}}$ a set of theory variables, $s, t \in \mathcal{T}(\mathcal{F}_{\mathsf{th}}, X)$, and φ a logical constraint over X. If $\models_{\mathcal{M}} (\varphi\sigma \Rightarrow s\sigma = t\sigma)$ holds for all X-valued substitutions σ, then $\mathcal{E} \models_{\mathsf{cec}} \Pi X.\, s \approx t\ [\varphi]$.*

Proof (Sketch). For any X-valued substitution σ with $\models_{\mathcal{M}} \varphi\sigma$, we have $[\![s\sigma]\!]_{\mathcal{M}} = [\![t\sigma]\!]_{\mathcal{M}}$. Then, use Lemma 2.3 to obtain $s\sigma \overset{*}{\leftrightarrow}_{\mathcal{E}} t\sigma$. ◀

▶ **Corollary 3.16.** *Let $\langle \mathcal{M}, \mathcal{E} \rangle$ be a CE-theory, $X \subseteq \mathcal{V}_{\mathsf{th}}$ a set of theory variables, and $\varphi \Rightarrow s = t$ a logical constraint over X such that $\models_{\mathcal{M}} (\varphi \Rightarrow s = t)$. Then, $\mathcal{E} \models_{\mathsf{cec}} \Pi X.\, s \approx t\ [\varphi]$.*

3.3 Relations to Conversion of Rewrite Steps

In this subsection, we present characterizations of CE-validity from the perspective of logically constrained rewriting with respect to equations.

▶ **Theorem 3.17.** *For a CE-theory $\langle \mathcal{M}, \mathcal{E} \rangle$, $s \overset{*}{\leftrightarrow}_{\mathcal{E}} t$ if and only if $\mathcal{E} \models_{\mathsf{cec}} \Pi\varnothing.\, s \approx t\ [\mathsf{true}]$.*

Proof (Sketch). We have $\mathcal{E} \models_{\mathsf{cec}} \Pi\varnothing.\, s \approx t\ [\mathsf{true}]$ if and only if $s\sigma \overset{*}{\leftrightarrow}_{\mathcal{E}} t\sigma$ for any $\varnothing$-valued substitution σ such that $\sigma \models \mathsf{true}$ if and only if $s\sigma \overset{*}{\leftrightarrow}_{\mathcal{E}} t\sigma$ for any substitution σ. Thus, the claim follows by Lemma 3.7. ◀

We consider now the general case with a possibly non-empty set X of theory variables and a non-trivial constraint $\varphi \neq \mathsf{true}$ (also $\neg\varphi$) for the CE $\Pi X.\, s \approx t\ [\varphi]$. In this case, the following partial characterization can be made by using the notion of trivial CEs [17]. We can naturally extend the notion of trivial CEs in [17] to our setting as follows: a CE $\Pi X.\, s \approx t\ [\varphi]$ is said to be *trivial* if $s\sigma = t\sigma$ for any X-valued substitution σ such that $\models_{\mathcal{M}} \varphi\sigma$.

▶ **Theorem 3.18.** *Let $\langle \mathcal{M}, \mathcal{E} \rangle$ be a CE-theory, and $\Pi X.\, s \approx t\ [\varphi]$ a CE. Suppose $s \overset{*}{\leftrightarrow}_{\mathcal{E}} s'$ and $t \overset{*}{\leftrightarrow}_{\mathcal{E}} t'$ for some s', t' such that $\Pi X.\, s' \approx t'\ [\varphi]$ is trivial. Then, $\mathcal{E} \models_{\mathsf{cec}} \Pi X.\, s \approx t\ [\varphi]$.*

Proof (Sketch). Take an arbitrary X-valued substitution σ such that $\models_{\mathcal{M}} \varphi\sigma$. Then, it follows from our assumptions that $s\sigma \overset{*}{\leftrightarrow}_{\mathcal{E}} s'\sigma = t'\sigma \overset{*}{\leftrightarrow}_{\mathcal{E}} t\sigma$. ◀

Unfortunately, none of the CE-consequences in Example 3.10 can be handled by Theorems 3.17 and 3.18.

4 Proving CE-Validity

As mentioned in the previous section, CE-validity of a CE $\Pi X.\, s \approx t\ [\varphi]$ with respect to a CE-theory $\langle \mathcal{M}, \mathcal{E} \rangle$ is very tedious to be shown by the convertibility of s and t in $\mathcal{E}$. This motivates us to introduce another approach to reason about CE-validity. In this section, we introduce an inference system for proving CE-validity of CEs together with a discussion on its applicability and generality.

4.1 Inference System $\mathbf{CEC}_0$ and Its Soundness

In this subsection, we introduce an inference system $\mathbf{CEC}_0$ (*Constrained Equational Calculus for elementary steps*) for proving CE-validity of CEs. We prove soundness of it, by which it is guaranteed that all CEs $\Pi X.\, s \approx t\ [\varphi]$ derivable from $\mathcal{E}$ in the system $\mathbf{CEC}_0$ are valid, i.e. $\mathcal{E} \models_{\mathsf{cec}} \Pi X.\, s \approx t\ [\varphi]$.

▶ **Definition 4.1** (derivation of $\mathbf{CEC}_0$). *Let $\mathfrak{T} = \langle \mathcal{M}, \mathcal{E} \rangle$ be a CE-theory. The inference system $\mathbf{CEC}_0$ consists of the inference rules given in Figure 2. We assume in the rules that X, Y range over a (possibly infinite) set of theory variables, φ ranges over logical constraints. Let $\Pi X.\, s \approx t\ [\varphi]$ be a CE. We say that $\Pi X.\, s \approx t\ [\varphi]$ is derivable in $\mathbf{CEC}_0$ from $\mathcal{E}$ (or $\Pi X.\, s \approx t\ [\varphi]$ is a consequence of $\mathcal{E}$), written by $\mathcal{E} \vdash_{\mathbf{CEC}_0} \Pi X.\, s \approx t\ [\varphi]$, if there exists a derivation of $\mathcal{E} \vdash \Pi X.\, s \approx t\ [\varphi]$ in the system $\mathbf{CEC}_0$. When no confusion arises, $\mathcal{E} \vdash_{\mathbf{CEC}_0} \Pi X.\, s \approx t\ [\varphi]$ is abbreviated as $\mathcal{E} \vdash \Pi X.\, s \approx t\ [\varphi]$.*

We proceed with intuitive explanations of each rule. The rules *Refl*, *Trans*, *Sym*, *Cong*, and *Rule* are counterparts of the inference rules used in equational logic.

In order to handle instantiations, we consider two cases, namely *Theory Instance* and *General Instance*. The former rule covers instantiations affecting the logical constraint whereas the latter covers the case not affecting it.

The *Weakening* and *Split* rules handle logical reasoning in constraints. The *Weakening* rule logically weakens the constraint equation by strengthening its constraint. Note here the direction of the entailment $\varphi \Rightarrow \psi$ in the side condition: the rule is sound because the constrained equation is valid under the stronger constraint φ if the equation is valid under the weaker constraint ψ. Since some rules, like *Cong* and *Trans*, have premises with equality constraints, it may be required to first apply the *Weakening* rule to synchronize the constraints before using these rules. On the other hand, in the *Split* rule, the constraint of the conclusion $\varphi \vee \psi$ is logically weaker than the independent ones, φ and ψ, in each premise. The inference is still sound as it only joins two premises. Using the *Split* rule, one can perform reasoning based on case analysis.

The *Axiom* rule makes it possible to use equational consequences entailed in the constraint part of equational reasoning. The *Abst* rule incorporates consequences entailed in the constraint part in a different way, that is, to infer a possible abstraction of the equation.

The rules *Enlarge* and *Restrict* are used to adjust the set of instantiated variables (the "ΠX" part of CEs), with the proviso that it does not affect the validity. Note that, in *Enlarge*, $Y \subseteq X$ implies the side condition $\mathcal{V}\mathsf{ar}(s,t) \cap (Y \setminus X) = \varnothing$. We also want to remark that despite its name, the restriction $X \subseteq Y$ can be added to *Enlarge*, provided that removed variables are not used in s, t (the side condition $\mathcal{V}\mathsf{ar}(s,t) \cap (Y \setminus X) = \varnothing$ has to be satisfied).

▶ **Lemma 4.2.** *Let $\langle \mathcal{M}, \mathcal{E} \rangle$ be a CE-theory. If $\mathcal{E} \vdash_{\mathbf{CEC}_0} \Pi X.\, s \approx t\ [\varphi]$, then $\Pi X.\, s \approx t\ [\varphi]$ is a CE.*

$$\textit{Refl}\quad \frac{}{\mathcal{E} \vdash \Pi X.\, s \approx s\ [\varphi]}\ \mathcal{V}\mathsf{ar}(\varphi) \subseteq X \qquad \textit{Trans}\quad \frac{\mathcal{E} \vdash \Pi X.\, s \approx t\ [\varphi] \quad \mathcal{E} \vdash \Pi X.\, t \approx u\ [\varphi]}{\mathcal{E} \vdash \Pi X.\, s \approx u\ [\varphi]}$$

$$\textit{Sym}\quad \frac{\mathcal{E} \vdash \Pi X.\, t \approx s\ [\varphi]}{\mathcal{E} \vdash \Pi X.\, s \approx t\ [\varphi]} \qquad \textit{Cong}\quad \frac{\mathcal{E} \vdash \Pi X.\, s_1 \approx t_1\ [\varphi] \quad \dots \quad \mathcal{E} \vdash \Pi X.\, s_n \approx t_n\ [\varphi]}{\mathcal{E} \vdash \Pi X.\, f(s_1, \dots, s_n) \approx f(t_1, \dots, t_n)\ [\varphi]}$$

$$\textit{Rule}\quad \frac{}{\mathcal{E} \vdash \Pi X.\, \ell \approx r\ [\varphi]}\ (\Pi X.\, \ell \approx r\ [\varphi]) \in \mathcal{E}$$

$$\textit{Theory Instance}\quad \frac{\mathcal{E} \vdash \Pi Y.\, s \approx t\ [\varphi]}{\mathcal{E} \vdash \Pi X.\, s\sigma \approx t\sigma\ [\varphi\sigma]}\ \forall x \in Y.\ x\sigma \in \mathcal{T}(\mathcal{F}_{\mathsf{th}}, X)$$

$$\textit{General Instance}\quad \frac{\mathcal{E} \vdash \Pi X.\, s \approx t\ [\varphi]}{\mathcal{E} \vdash \Pi X.\, s\sigma \approx t\sigma\ [\varphi]}\ \mathcal{D}\mathsf{om}(\sigma) \cap X = \varnothing$$

$$\textit{Weakening}\quad \frac{\mathcal{E} \vdash \Pi X.\, s \approx t\ [\psi]}{\mathcal{E} \vdash \Pi X.\, s \approx t\ [\varphi]}\ \models_{\mathcal{M}} (\varphi \Rightarrow \psi), \mathcal{V}\mathsf{ar}(\varphi) \subseteq X$$

$$\textit{Split}\quad \frac{\mathcal{E} \vdash \Pi X.\, s \approx t\ [\varphi] \quad \mathcal{E} \vdash \Pi X.\, s \approx t\ [\psi]}{\mathcal{E} \vdash \Pi X.\, s \approx t\ [\varphi \vee \psi]}$$

$$\textit{Axiom}\quad \frac{}{\mathcal{E} \vdash \Pi X.\, s \approx t\ [\varphi]}\ \begin{array}{l}\models_{\mathcal{M}} (\varphi\sigma \Rightarrow s\sigma = t\sigma) \text{ for all } \sigma \text{ s.t. } X \subseteq \mathcal{V}\mathcal{D}\mathsf{om}(\sigma),\\ \mathcal{V}\mathsf{ar}(\varphi) \subseteq X\end{array}$$

$$\textit{Abst}\quad \frac{\mathcal{E} \vdash \Pi X.\, s\sigma \approx t\sigma\ [\varphi\sigma]}{\mathcal{E} \vdash \Pi X.\, s \approx t\ [\varphi]}\ \begin{array}{l}\models_{\mathcal{M}} (\varphi \Rightarrow \bigwedge_{x \in X} x = \sigma(x)),\\ \mathcal{V}\mathsf{ar}(\varphi) \subseteq X, \left(\bigcup_{x \in X} \mathcal{V}\mathsf{ar}(\sigma(x))\right) \subseteq X\end{array}$$

$$\textit{Enlarge}\quad \frac{\mathcal{E} \vdash \Pi Y.\, s \approx t\ [\varphi]}{\mathcal{E} \vdash \Pi X.\, s \approx t\ [\varphi]}\ \mathcal{V}\mathsf{ar}(s, t) \cap (Y \setminus X) = \varnothing,\ \mathcal{V}\mathsf{ar}(\varphi) \subseteq X$$

Figure 2 Inference rules of $\mathbf{CEC}_0$.

Proof (Sketch). The proof proceeds by induction on the derivation of $\mathcal{E} \vdash \Pi X.\, s \approx t\ [\varphi]$ that (1) $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ have the same sort, (2) φ is a logical constraint, (3) $X \subseteq \mathcal{V}_{\mathsf{th}}$, and (4) $\mathcal{V}\mathsf{ar}(\varphi) \subseteq X$. ◀

Below we present some examples of derivations which cover all of our inference rules at least once. In the following example we denote *Theory Instance* by *TInst* and *General Instance* by *GInst* accordingly.

▶ **Example 4.3.** Let $\langle \mathcal{M}, \mathcal{E} \rangle$ be the CE-theory given in Example 3.5. Below, we present a derivation of $\mathcal{E} \vdash \Pi\{n\}.\, \mathsf{exp}(x, n) * \mathsf{exp}(x, -n) \approx \mathsf{e}$.

$$\dfrac{\dfrac{\dfrac{}{\mathcal{E} \vdash \Pi\{n, m\}.\, \mathsf{exp}(x, n) * \mathsf{exp}(x, m) \approx \mathsf{exp}(x, n + m)}\ \textit{Rule}}{\mathcal{E} \vdash \Pi\{n\}.\, \mathsf{exp}(x, n) * \mathsf{exp}(x, -n) \approx \mathsf{exp}(x, n + (-n))}\ \textit{TInst} \qquad (1)}{\mathcal{E} \vdash \Pi\{n\}.\, \mathsf{exp}(x, n) * \mathsf{exp}(x, -n) \approx \mathsf{e}}\ \textit{Trans}$$

where (1) is

$$\dfrac{\dfrac{\dfrac{}{\mathcal{E} \vdash x \approx x}\,\mathit{Refl}}{\mathcal{E} \vdash \Pi\{n\}.\, x \approx x}\,\mathit{Enlarge} \quad \dfrac{}{\mathcal{E} \vdash \Pi\{n\}.\, n + (-n) \approx 0}\,\mathit{Axiom}}{\dfrac{\mathcal{E} \vdash \Pi\{n\}.\, \mathsf{exp}(x, n + (-n)) \approx \mathsf{exp}(x, 0)}{\mathcal{E} \vdash \Pi\{n\}.\, \mathsf{exp}(x, n + (-n)) \approx \mathsf{e}}\,\mathit{Trans}}\,\mathit{Cong} \quad \dfrac{\dfrac{}{\mathcal{E} \vdash \mathsf{exp}(x, 0) \approx \mathsf{e}}\,\mathit{Rule}}{\mathcal{E} \vdash \Pi\{n\}.\, \mathsf{exp}(x, 0) \approx \mathsf{e}}\,\mathit{Enlarge}$$

Here, $\mathcal{E} \vdash \Pi\{n\}.\, n + (-n) \approx 0$ is derived by the *Axiom* rule, because of $\models_{\mathcal{M}} \sigma(n + (-n)) = 0$ holds for all $\{n\}$-valued substitutions σ.

▶ **Example 4.4.** Let $\langle \mathcal{M}, \mathcal{E} \rangle$ be the CE-theory given in Example 3.6. Below, we present a derivation of $\mathcal{E} \vdash \Pi\{n\}.\, \mathsf{nth}(x{::}y{::}zs, n + 2) \approx \mathsf{nth}(zs, n)\ [n > 0]$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathcal{E} \vdash \Pi\{n\}.\, \mathsf{nth}(x{::}xs, n) \approx \mathsf{nth}(xs, n-1)\ [n > 0]}\,\mathit{Rule}}{\mathcal{E} \vdash \Pi\{n\}.\, \mathsf{nth}(x{::}xs, n+1) \approx \mathsf{nth}(xs, (n+1)-1)\ [n+1 > 0]}\,\mathit{TInst} \quad (1)}{\mathcal{E} \vdash \Pi\{n\}.\, \mathsf{nth}(x{::}xs, n+1) \approx \mathsf{nth}(xs, n)\ [n+1 > 0]}\,\mathit{Trans}}{\mathcal{E} \vdash \Pi\{n\}.\, \mathsf{nth}(x{::}xs, n+1) \approx \mathsf{nth}(xs, n)\ [n > 0]}\,\mathit{Weaken} \quad \dfrac{}{\mathcal{E} \vdash \Pi\{n\}.\, \mathsf{nth}(y{::}zs, n+1) \approx \mathsf{nth}(zs, n)\ [n > 0]}\,\mathit{GInst}}{\mathcal{E} \vdash \Pi\{n\}.\, \mathsf{nth}(x{::}y{::}zs, n+2) \approx \mathsf{nth}(zs, n)\ [n > 0]}\,\mathit{Trans}\ (2)$$

where (1) is

$$\dfrac{\dfrac{\dfrac{}{\mathcal{E} \vdash \Pi\{n\}.\, xs \approx xs}\,\mathit{Refl} \quad \dfrac{}{\mathcal{E} \vdash \Pi\{n\}.\, (n+1)-1 \approx n}\,\mathit{Axiom}}{\mathcal{E} \vdash \Pi\{n\}.\, \mathsf{nth}(xs, ((n+1)-1) \approx \mathsf{nth}(xs, n)}\,\mathit{Cong}}{\mathcal{E} \vdash \Pi\{n\}.\, \mathsf{nth}(xs, ((n+1)-1) \approx \mathsf{nth}(xs, n)\ [n+1 > 0]}\,\mathit{Weaken}$$

and (2) is

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathcal{E} \vdash \Pi\{n\}.\, \mathsf{nth}(x :: xs, n) \approx \mathsf{nth}(xs, n-1)\ [n > 0]}\,\mathit{Rule}}{\mathcal{E} \vdash \Pi\{n\}.\, \mathsf{nth}(x :: xs, n+2) \approx \mathsf{nth}(xs, (n+2)-1)\ [n+2 > 0]}\,\mathit{TInst} \quad (3)}{\mathcal{E} \vdash \Pi\{n\}.\, \mathsf{nth}(x :: xs, n+2) \approx \mathsf{nth}(xs, n+1)\ [n+2 > 0]}\,\mathit{Trans}}{\mathcal{E} \vdash \Pi\{n\}.\, \mathsf{nth}(x :: ys, n+2) \approx \mathsf{nth}(ys, n+1)\ [n > 0]}\,\mathit{Weaken}}{\mathcal{E} \vdash \Pi\{n\}.\, \mathsf{nth}(x :: y :: zs, n+2) \approx \mathsf{nth}(y :: zs, n+1)\ [n > 0]}\,\mathit{GInst}$$

where (3) is

$$\dfrac{\dfrac{\dfrac{}{\mathcal{E} \vdash \Pi\{n\}.\, xs \approx xs}\,\mathit{Refl} \quad \dfrac{}{\mathcal{E} \vdash \Pi\{n\}.\, (n+2)-1 \approx n+1}\,\mathit{Axiom}}{\mathcal{E} \vdash \Pi\{n\}.\, \mathsf{nth}(xs, ((n+2)-1) \approx \mathsf{nth}(xs, n+1)}\,\mathit{Cong}}{\mathcal{E} \vdash \Pi\{n\}.\, \mathsf{nth}(xs, (n+2)-1) \approx \mathsf{nth}(xs, n+1)\ [n+2 > 0]}\,\mathit{Weaken}$$

In the next example, we illustrate usages of the *Split* rule and the *Abst* rule.

▶ **Example 4.5.** Let $\mathsf{f} \colon \mathsf{Bool} \to \mathsf{Int} \in \mathcal{F}_{\mathsf{te}}$. Let $\mathcal{E} = \{\Pi\varnothing.\, \mathsf{f}(\mathsf{true}) \approx 0\ [\mathsf{true}], \Pi\varnothing.\, \mathsf{f}(\mathsf{false}) \approx 0\ [\mathsf{true}]\}$. Then we have $\mathsf{f}(\mathsf{true}) \leftrightarrow_{\mathcal{E}} 0$ and $\mathsf{f}(\mathsf{false}) \leftrightarrow_{\mathcal{E}} 0$. Thus, for all $\{x\}$-valued substitutions σ, we have $\mathsf{f}(x)\sigma \overset{*}{\leftrightarrow}_{\mathcal{E}} 0\sigma$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathcal{E} \vdash \Pi\varnothing.\, \mathsf{f}(\mathsf{true}) \approx 0}\,\mathit{Rule}}{\mathcal{E} \vdash \Pi\varnothing.\, \mathsf{f}(\mathsf{true}) \approx 0\ [\mathsf{true} = \mathsf{true}]}\,\mathit{Weaken}}{\mathcal{E} \vdash \Pi\{x\}.\, \mathsf{f}(x) \approx 0\ [x = \mathsf{true}]}\,\mathit{Abst} \quad \dfrac{\dfrac{\dfrac{}{\mathcal{E} \vdash \Pi\varnothing.\, \mathsf{f}(\mathsf{false}) \approx 0}\,\mathit{Rule}}{\mathcal{E} \vdash \Pi\varnothing.\, \mathsf{f}(\mathsf{false}) \approx 0\ [\mathsf{false} = \mathsf{false}]}\,\mathit{Weaken}}{\mathcal{E} \vdash \Pi\{x\}.\, \mathsf{f}(x) \approx 0\ [x = \mathsf{false}]}\,\mathit{Abst}}{\mathcal{E} \vdash \Pi\{x\}.\, \mathsf{f}(x) \approx 0\ [x = \mathsf{true} \vee x = \mathsf{false}]}\,\mathit{Split}}{\mathcal{E} \vdash \Pi\{x\}.\, \mathsf{f}(x) \approx 0\ [\mathsf{true}]}\,\mathit{Weaken}$$

We now present soundness of the system $\mathbf{CEC}_0$ with respect to CE-validity.

▶ **Theorem 4.6** (soundness of the system $\mathbf{CEC}_0$)**.** *Let $\langle \mathcal{M}, \mathcal{E} \rangle$ be a CE-theory and $\Pi X.\, s \approx t\ [\varphi]$ a CE. If $\mathcal{E} \vdash_{\mathbf{CEC}_0} \Pi X.\, s \approx t\ [\varphi]$, then $\mathcal{E} \models_{\mathsf{cec}} \Pi X.\, s \approx t\ [\varphi]$.*

Proof (Sketch). The proof proceeds by induction on the derivation. The cases for all rules except for *Abst* and *Enlarge* follow by Lemmas 3.11–3.15 and well-known properties in logic. For the case *Abst*, suppose that $\mathcal{E} \vdash \Pi X.\, s \approx t\ [\varphi]$ is derived from $\mathcal{E} \vdash \Pi X.\, s\sigma \approx t\sigma\ [\varphi\sigma]$ as given in Figure 2. Let ρ be an X-valued substitution such that $\models_{\mathcal{M}} \varphi\rho$. Then by the side conditions we have $[\![\rho(x)]\!] = [\![\rho(\sigma(x))]\!]$ for any $x \in X$. Hence, $\models_{\mathcal{M}} \varphi\sigma\rho$, and $s\sigma\rho \overset{*}{\leftrightarrow}_{\mathcal{E}} t\sigma\rho$ by the induction hypothesis. Then by $\mathcal{V}\mathsf{ar}(s,t) \subseteq X$, we have $s\rho \overset{*}{\leftrightarrow}_{\mathcal{E}} s\sigma\rho \overset{*}{\leftrightarrow}_{\mathcal{E}} t\sigma\rho \overset{*}{\leftrightarrow}_{\mathcal{E}} t\rho$. For the case *Enlarge*, suppose that $\mathcal{E} \vdash \Pi X.\, s \approx t\ [\varphi]$ is derived from $\mathcal{E} \vdash \Pi Y.\, s \approx t\ [\varphi]$ as given in Figure 2. Let δ be an X-valued substitution such that $\models_{\mathcal{M}} \varphi\delta$. Define a substitution δ' as follows: $\delta'(z^\tau) = c^\tau$ if $x \in Y \setminus X$, and $\delta'(x) = \delta(x)$ otherwise, where c^τ is (arbitrarily) taken from $\mathcal{V}\mathsf{al}^\tau$. Clearly, δ' is Y-valued. We also have $s\delta' = s\delta$, $t\delta' = t\delta$, and $\varphi\delta' = \varphi\delta$ by the side conditions. Thus, $s\delta = s\delta' \overset{*}{\leftrightarrow}_{\mathcal{E}} t\delta' = t\delta$ using the induction hypothesis. ◀

▶ Remark 4.7. We remark that some of our inference rules utilize validity in the model. Thus, $\mathbf{CEC}_0$ does not have convenient properties like recursive enumerability of its theorems like we are used to from other formal systems.

4.2 Partial Completeness of $\mathbf{CEC}_0$

In this subsection, we present some results regarding the completeness property of $\mathbf{CEC}_0$.

▶ **Lemma 4.8.** *Let $\langle \mathcal{M}, \mathcal{E} \rangle$ be a CE-theory and $\Pi X.\, s \approx t\ [\varphi]$ a CE such that φ is satisfiable. Suppose $s\sigma = t\sigma$ for all X-valued substitutions σ such that $\models_{\mathcal{M}} \varphi\sigma$. Then $\mathcal{E} \vdash \Pi X.\, s \approx t\ [\varphi]$.*

Proof (Sketch). The case $s, t \in \mathcal{T}(\mathcal{F}_{\mathsf{th}}, X)$ follows by the assumption using the *Axiom* rule. Next, we consider the case $s = x \in \mathcal{V}$ with $x \notin X$. In this case, we can derive $t = x$ using the assumption, and the case follows using the *Refl* rule. For the general case, from the assumption, one can let $s = C[s_1, \ldots, s_n]$ and $t = C[t_1, \ldots, t_n]$ for a multi-hole context C and terms s_i, t_i $(1 \leqslant i \leqslant n)$ such that either one of s_i or t_i is a variable. Thus, by the previous cases, we know that $\mathcal{E} \vdash \Pi X.\, s_i \approx t_i\ [\varphi]$ for each $1 \leqslant i \leqslant n$, possibly using the *Sym* rule. Thus, the claim is obtained using *Refl*, *Trans* and *Cong* rules. ◀

▶ **Lemma 4.9.** *Let $\langle \mathcal{M}, \mathcal{E} \rangle$ be a CE-theory and $\Pi X.\, s \approx t\ [\varphi]$ a CE such that φ is satisfiable. Suppose $s\sigma \leftrightarrow^{=}_{\mathsf{calc}} t\sigma$ for all X-valued substitutions σ such that $\mathcal{D}\mathsf{om}(\sigma) = X$ and $\models_{\mathcal{M}} \varphi\sigma$. Then $\mathcal{E} \vdash \Pi X.\, s \approx t\ [\varphi]$.*

Proof (Sketch). First of all, the claim for the case $s, t \in \mathcal{T}(\mathcal{F}_{\mathsf{th}}, X)$ follows using the *Axiom* rule. If $s\sigma = t\sigma$ for all X-valued substitutions σ, the claim follows from Lemma 4.8. Thus, it remains to consider the case that there exists an X-valued substitution σ such that $s\sigma \leftrightarrow_{\mathsf{calc}} t\sigma$, $\models_{\mathcal{M}} \varphi\sigma$ and $X = \mathcal{D}\mathsf{om}(\sigma)$. Suppose that $s\sigma = C[f(v_1, \ldots, v_n)]_q$ and $t\sigma = C[v_0]_q$ with $f \in \mathcal{F}_{\mathsf{th}}$ and $v_0, \ldots, v_n \in \mathcal{V}\mathsf{al}$ such that $\mathcal{I}(f)(v_1, \ldots, v_n) = v_0$. As $\mathcal{D}\mathsf{om}(\sigma) = X$ and σ is X-valued, we have $s|_q = f(s_1, \ldots, s_n)$ with $s_1, \ldots, s_n \in \mathcal{V}\mathsf{al} \cup X$ and $t|_q \in \mathcal{V}\mathsf{al} \cup X$; hence $s|_q, t|_q \in \mathcal{T}(\mathcal{F}_{\mathsf{th}}, X)$, and thus, $\mathcal{E} \vdash \Pi X.\, s|_q \approx t|_q\ [\varphi]$ holds as we mentioned above. Let $s = C_1[f(s_1, \ldots, s_n)]_q$ and $t = C_2[t_0]_q$ with $C_1\sigma = C_2\sigma$ for any X-valued substitution σ such that $\mathcal{D}\mathsf{om}(\sigma) = X$ and $\models_{\mathcal{M}} \varphi\sigma$. Then, $\mathcal{E} \vdash \Pi X.\, s \approx t\ [\varphi]$ follows using $\mathcal{E} \vdash \Pi X.\, s|_q \approx t|_q\ [\varphi]$, Lemma 4.8, and the *Cong* and *Trans* rules. ◀

From Lemma 4.9, the partial completeness for at most one calculation step follows.

▶ **Theorem 4.10.** *Let $\langle \mathcal{M}, \mathcal{E} \rangle$ be a CE-theory and $\Pi X.\, s \approx t\ [\varphi]$ a CE such that φ is satisfiable. Suppose $s\sigma \leftrightarrow^{=}_{\mathsf{calc}} t\sigma$ for all X-valued substitutions σ such that $\models_{\mathcal{M}} \varphi\sigma$. Then $\mathcal{E} \vdash_{\mathbf{CEC}_0} \Pi X.\, s \approx t\ [\varphi]$.*

One may expect that Theorem 4.10 can be extended to the general completeness theorem for arbitrary $\mathcal{E}$ in such a way that $\mathcal{E} \models_{\mathsf{cec}} \Pi X.\, s \approx t\ [\varphi]$ implies $\mathcal{E} \vdash \Pi X.\, s \approx t\ [\varphi]$ (full completeness). Rephrasing this, we have: if $s\sigma \overset{*}{\leftrightarrow}_{\mathcal{E}} t\sigma$ for all X-valued substitutions σ such that $\models_{\mathcal{M}} \varphi\sigma$, then $\mathcal{E} \vdash \Pi X.\, s \approx t\ [\varphi]$. The partial completeness result above is far from this formulation of full completeness in that the assumption does not assume arbitrary conversions $s\sigma \overset{*}{\leftrightarrow}_{\mathcal{E}} t\sigma$ but only $s\sigma \leftrightarrow^{=}_{\mathsf{calc}} t\sigma$ (i.e. at most one calculation step). However, full completeness does not seem to hold for the system $\mathbf{CEC}_0$, as witnessed by the following example.

▶ **Example 4.11.** Consider the following LCES:

$$\mathcal{E} = \begin{cases} \Pi\{x\}.\, \mathsf{nneg}(x) \approx \mathsf{true}\ [x = 0] & (1) \\ \Pi\{x, y\}.\, \mathsf{nneg}(x) \approx \mathsf{nneg}(y)\ [x + 1 = y] & (2) \end{cases}$$

For each $n \geqslant 0$, we have $\mathsf{nneg}(n) \overset{*}{\leftrightarrow}_{\mathcal{E}} \mathsf{true}$:

$$\mathsf{nneg}(n) \leftrightarrow_{\mathcal{E}} \mathsf{nneg}(n-1) \leftrightarrow_{\mathcal{E}} \cdots \leftrightarrow_{\mathcal{E}} \mathsf{nneg}(0) \leftrightarrow_{\mathcal{E}} \mathsf{true}$$

Thus, for the CE $\Pi\{x\}.\, \mathsf{nneg}(x) \approx \mathsf{true}\ [x \geqslant 0]$, we have for all σ such that $\models_{\mathcal{M}} \sigma(x) \geqslant 0$, $\mathsf{nneg}(x)\sigma \overset{*}{\leftrightarrow}_{\mathcal{E}} \mathsf{true}\sigma = \mathsf{true}$. On the other hand, for each $n \geqslant 0$, one can give a derivation of $\Pi\varnothing.\, \mathsf{nneg}(n) \approx \mathsf{true}$ – for example, for $n = 2$,

$$\dfrac{\dfrac{\dfrac{}{(2)}\ \mathit{Rule}}{\Pi\varnothing.\, \mathsf{nneg}(2) \approx \mathsf{nneg}(1)}\ \mathit{TInst} \quad \dfrac{\dfrac{\dfrac{}{(2)}\ \mathit{Rule}}{\Pi\varnothing.\, \mathsf{nneg}(1) \approx \mathsf{nneg}(0)}\ I \quad \dfrac{\dfrac{}{(1)}\ \mathit{Rule}}{\Pi\varnothing.\, \mathsf{nneg}(0) \approx \mathsf{true}}\ \mathit{TInst}}{\Pi\varnothing.\, \mathsf{nneg}(1) \approx \mathsf{true}}\ \mathit{Trans}}{\Pi\varnothing.\, \mathsf{nneg}(2) \approx \mathsf{true}}\ \mathit{Trans}$$

However, these derivations differ for each $n \geqslant 0$, and and are hardly merged. As a conclusion, it seems that the CE $\Pi\{x\}.\, \mathsf{nneg}(x) \approx \mathsf{true}\ [x \geqslant 0]$ is beyond the derivability of $\mathbf{CEC}_0$.

▶ Remark 4.12. After looking at Example 4.11, it might seem reasonable that adding some kind of induction reasoning is required for our proof system. However, rules for the induction on positive integers, etc. are only possible when working with a specific model. Such rules have clearly a different nature than rules in our calculus that work with any underlying model. Our calculus $\mathbf{CEC}_0$ intends to be a general calculus that is free from specific underlying models and does not include model-specific rules.

▶ Remark 4.13. We remark a difficulty to extend Theorem 4.10 to multiple (calculation) steps, i.e. to have a statement like $\mathcal{E} \vdash_{\mathbf{CEC}_0} \Pi X.\, s \approx t\ [\varphi]$ whenever $s\sigma \leftrightarrow^{*}_{\mathsf{calc}} t\sigma$ for all X-valued substitutions σ such that $\models_{\mathcal{M}} \varphi\sigma$. Or even to obtain a slightly weaker statement like $\mathcal{E} \vdash_{\mathbf{CEC}_0} \Pi X.\, s \approx t\ [\varphi]$ whenever there exists a natural number n such that $s\sigma \leftrightarrow^{n}_{\mathsf{calc}} t\sigma$ for all X-valued substitutions σ such that $\models_{\mathcal{M}} \varphi\sigma$. It might look that induction on the length of $s\sigma \leftrightarrow^{*}_{\mathsf{calc}} t\sigma$ (or the one on n in the case of the latter) can be applied. However, to apply Theorem 4.10 to each step, we need the form $s_0\sigma \leftrightarrow_{\mathsf{calc}} s_1\sigma \leftrightarrow_{\mathsf{calc}} \cdots \leftrightarrow_{\mathsf{calc}} s_n\sigma$ for each σ which is not generally implied by $s\sigma \leftrightarrow^{*}_{\mathsf{calc}} t\sigma$ or $s\sigma \leftrightarrow^{n}_{\mathsf{calc}} t\sigma$, as intermediate terms may vary depending on the substitution σ. Extending Theorem 4.10 to multiple steps remains as our future work.

▶ Remark 4.14. Our final remark deals with the difficulty to extend Theorem 4.10 to a single rule step, i.e. to have a statement like $\mathcal{E} \vdash_{\mathbf{CEC}_0} \Pi X.\, s \approx t\ [\varphi]$ whenever $s\sigma \leftrightarrow_{\mathsf{rule},\mathcal{E}} t\sigma$ for all X-valued substitutions σ such that $\models_{\mathcal{M}} \varphi\sigma$. For the rule step, there may be multiple choices of positions and multiple choices of CEs to be applied for the step $s\sigma \leftrightarrow_{\mathsf{rule},\mathcal{E}} t\sigma$. Thus, we have to divide X-valued substitutions satisfying φ depending on each position p that a CE is applied and each applied CE $\Pi X_i.\, \ell_i \approx r_i\ [\varphi_i] \in \mathcal{E}$, and combine the obtained consequences. However, it is in general not guaranteed that such a division of substitutions can be characterized by a constraint. Note that the set of sets of substitutions is in general not countable but the set of constraints is countable. Thus, it may be necessary to assume some assumption on the expressiveness of constraints to obtain the extension for the single rule step. On the other hand, we conjecture that the (full) completeness would hold for CE-theories with a finite underlying model.

5 Algebraic Semantics for CE-Validity

In this section, we explore algebraic semantics for CE-validity. In this approach, CE-validity is characterized by validity in models in a class of algebras, which we call *CE-algebras*. We show that this characterization is sound and complete, in the sense that CE-validity can be fully characterized.

5.1 CE-Algebras

In this subsection, we introduce a notion of CE-algebras and validity in them. After presenting basic properties of our semantics, we prove its soundness with respect to the CE-validity.

▶ **Definition 5.1** (CE-$\langle\Sigma,\mathcal{M}\rangle$-algebra). *Let $\Sigma = \langle \mathcal{S}_{\mathsf{th}}, \mathcal{S}_{\mathsf{te}}, \mathcal{F}_{\mathsf{th}}, \mathcal{F}_{\mathsf{te}}\rangle$ be a signature and $\mathcal{M} = \langle \mathcal{I}, \mathcal{J}\rangle$ be a model over $\mathcal{S}_{\mathsf{th}}$ and $\mathcal{F}_{\mathsf{th}}$. A* constrained equational $\langle\Sigma,\mathcal{M}\rangle$-algebra *(CE-$\langle\Sigma,\mathcal{M}\rangle$-algebra, for short) is a pair $\mathfrak{M} = \langle \mathfrak{I}, \mathfrak{J}\rangle$ where $\mathfrak{I}$ assigns each $\tau \in \mathcal{S}$ a non-empty set $\mathfrak{I}(\tau)$, specifying its domain, and $\mathfrak{J}$ assigns each $f\colon \tau_1 \times \cdots \times \tau_n \to \tau_0 \in \mathcal{F}$ an interpretation function $\mathfrak{J}(f)\colon \mathfrak{I}(\tau_1) \times \cdots \times \mathfrak{I}(\tau_n) \to \mathfrak{I}(\tau_0)$ that extends the model $\mathcal{M} = \langle \mathcal{I}, \mathcal{J}\rangle$, that is, $\mathfrak{I}(\tau) \supseteq \mathcal{I}(\tau)$ for all $\tau \in \mathcal{S}_{\mathsf{th}}$ and $\mathfrak{J}(f) \restriction_{\mathcal{I}(\tau_1)\times\cdots\times\mathcal{I}(\tau_n)} = \mathcal{J}(f)$ for all $f \in \mathcal{F}_{\mathsf{th}}$ (or more generally there exists an injective homomorphism $\iota\colon \mathcal{M} \to \mathfrak{M}$).*

Let $\mathfrak{M} = \langle \mathfrak{I}, \mathfrak{J}\rangle$ be a CE-$\langle\Sigma,\mathcal{M}\rangle$-algebra. A valuation ρ over $\mathfrak{M}$ is defined similarly to $\mathcal{M}$, but $\mathcal{S}$ instead of $\mathcal{S}_{\mathsf{th}}$, $\mathcal{F}$ instead of $\mathcal{F}_{\mathsf{th}}$, etc. Similarly, a valuation ρ over $\mathfrak{M}$ satisfies a logical constraint φ, denoted by $\models_{\mathfrak{M}} \varphi$, if $[\![\varphi]\!]_{\rho,\mathfrak{M}} = \mathsf{true}$.

Careful readers may wonder why the interpretation functions for the theory part are not the same but an extension of the underlying model $\mathcal{M} = \langle \mathcal{I}, \mathcal{J}\rangle$. Indeed, in the definition of CE-$\langle\Sigma,\mathcal{M}\rangle$-algebras $\mathfrak{M} = \langle \mathfrak{I}, \mathfrak{J}\rangle$ above, we only demand that $\mathfrak{I}(\tau) \supseteq \mathcal{I}(\tau)$ for all $\tau \in \mathcal{S}_{\mathsf{th}}$ and not $\mathfrak{I}(\tau) = \mathcal{I}(\tau)$ for all $\tau \in \mathcal{S}_{\mathsf{th}}$. In fact, this is required to obtain the completeness result; however, this explanation is postponed until Example 5.18. We continue to present some basic properties of our semantics which are proven in a straightforward manner.

▶ **Lemma 5.2.** *Let $\mathfrak{T} = \langle \mathcal{M}, \mathcal{E}\rangle$ be a CE-theory over a signature Σ, and $\mathfrak{M} = \langle \mathfrak{I}, \mathfrak{J}\rangle$ a CE-$\langle\Sigma,\mathcal{M}\rangle$-algebra. Then, the binary relation over terms given by $[\![\cdot]\!]_{\mathfrak{M},\rho} = [\![\cdot]\!]_{\mathfrak{M},\rho}$ for any valuation ρ on $\mathfrak{M}$, is closed under substitutions and contexts.*

▶ **Lemma 5.3.** *Let $\mathfrak{T} = \langle \mathcal{M}, \mathcal{E}\rangle$ be a CE- theory over a signature Σ such that $\mathcal{M} = \langle \mathcal{I}, \mathcal{J}\rangle$, $\mathfrak{M}$ a CE-$\langle\Sigma,\mathcal{M}\rangle$-algebra, and $X \subseteq \mathcal{V}_{\mathsf{th}}$ a set of theory variables and suppose $t \in \mathcal{T}(\mathcal{F}_{\mathsf{th}}, X)$. Then, for any valuation ρ on $\mathfrak{M}$ such that $\rho(x) \in \mathcal{I}(\tau)$ for all $x^\tau \in X$, we have $[\![t]\!]_{\mathfrak{M},\rho} = [\![t]\!]_{\mathcal{M},\rho}$.*

Next, we extend the definition of validity on CE-algebras to CEs, by which we can give a notion of models of CE-theories, and the semantic consequence relation.

▶ **Definition 5.4** (model of constrained equational theory). *Let $\mathfrak{T} = \langle \mathcal{M}, \mathcal{E} \rangle$ be a CE-theory over a signature Σ such that $\mathcal{M} = \langle \mathcal{I}, \mathcal{J} \rangle$, and $\mathfrak{M} = \langle \mathfrak{I}, \mathfrak{J} \rangle$ a CE-$\langle \Sigma, \mathcal{M} \rangle$-algebra.*

1. *A CE $\Pi X.\, \ell \approx r \ [\varphi]$ is said to be* valid *in $\mathfrak{M}$, denoted by $\models_{\mathfrak{M}} \Pi X.\, \ell \approx r \ [\varphi]$, if for all valuations ρ over $\mathfrak{M}$ satisfying the constraint φ (i.e. $[\![\varphi]\!]_{\mathfrak{M},\rho} = \mathsf{true}$ holds) and $\rho(x) \in \mathcal{I}(\tau)$ holds for all $x^\tau \in X$, we have $[\![\ell]\!]_{\mathfrak{M},\rho} = [\![r]\!]_{\mathfrak{M},\rho}$.*
2. *A CE-$\langle \Sigma, \mathcal{M} \rangle$-algebra $\mathfrak{M} = \langle \mathfrak{I}, \mathfrak{J} \rangle$ is said to be a* model *of the CE-theory $\mathfrak{T}$ if $\models_{\mathfrak{M}} \mathcal{E}$. Here, $\models_{\mathfrak{M}} \mathcal{E}$ denotes that $\models_{\mathfrak{M}} \Pi X.\, \ell \approx r \ [\varphi]$ for all $\Pi X.\, \ell \approx r \ [\varphi] \in \mathcal{E}$.*
3. *Let $\Pi X.\, \ell \approx r \ [\varphi]$ be a CE. We write $\mathfrak{T} \models \Pi X.\, \ell \approx r \ [\varphi]$ (or $\mathcal{E} \models \Pi X.\, \ell \approx r \ [\varphi]$ if no confusion arises) if $\models_{\mathfrak{M}} \Pi X.\, \ell \approx r \ [\varphi]$ holds for all CE-$\langle \Sigma, \mathcal{M} \rangle$-algebras $\mathfrak{M}$ that are models of $\mathfrak{T}$.*

We remark that, in item **1**, as $\varphi \in \mathcal{T}(\mathcal{F}_{\mathsf{th}}, \mathcal{V}_{\mathsf{th}})$, we have $[\![\varphi]\!]_{\mathfrak{M},\rho} = \mathsf{true}$ if and only if $[\![\varphi]\!]_{\mathcal{M},\rho} = \mathsf{true}$ by Lemma 5.3. Based on the preceding lemmas, soundness of our semantics with respect to conversion is not difficult to obtain.

▶ **Lemma 5.5** (soundness w.r.t. conversion). *Let $\mathfrak{T} = \langle \mathcal{M}, \mathcal{E} \rangle$ be a CE-theory over a signature Σ, and $\mathfrak{M} = \langle \mathfrak{I}, \mathfrak{J} \rangle$ a CE-$\langle \Sigma, \mathcal{M} \rangle$-algebra such that $\models_{\mathfrak{M}} \mathcal{E}$. If $s \overset{*}{\leftrightarrow}_{\mathcal{E}} t$ then $[\![s]\!]_{\mathfrak{M},\rho} = [\![t]\!]_{\mathfrak{M},\rho}$ for any valuation ρ on $\mathfrak{M}$.*

Proof (Sketch). It suffices to consider the case $s \leftrightarrow_{\mathcal{E}} t$ with a root step; the claim easily follows from Lemma 5.2. Let $\Sigma = \langle \mathcal{S}_{\mathsf{th}}, \mathcal{S}_{\mathsf{te}}, \mathcal{F}_{\mathsf{th}}, \mathcal{F}_{\mathsf{te}} \rangle$ and $\mathcal{M} = \langle \mathcal{I}, \mathcal{J} \rangle$. Let $s \leftrightarrow_{\mathsf{calc}} t$. Then, $s, t \in \mathcal{T}(\mathcal{F}_{\mathsf{th}})$, and hence $[\![s]\!]_{\mathcal{M}} = [\![t]\!]_{\mathcal{M}}$. Thus, $[\![s]\!]_{\mathfrak{M}} = [\![t]\!]_{\mathfrak{M}}$ by Lemma 5.3. Otherwise, let $s \leftrightarrow_{\mathsf{rule},\mathcal{E}} t$. Then, $s = \ell\sigma$ and $t = r\sigma$ for some $\Pi X.\, \ell \approx r \ [\varphi] \in \mathcal{E}$ and an X-valued substitution σ such that $\models_{\mathcal{M}} \varphi\sigma$. We have a valuation $[\![\sigma]\!]_{\mathfrak{M},\rho}$ on $\mathfrak{M}$ by $[\![\sigma]\!]_{\mathfrak{M},\rho}(y) = [\![\sigma(y)]\!]_{\mathfrak{M},\rho}$ for any $y \in \mathcal{V}$. Then, similarly to Lemma 2.1, we have $[\![u\sigma]\!]_{\mathfrak{M},\rho} = [\![u]\!]_{\mathfrak{M},[\![\sigma]\!]_{\mathfrak{M},\rho}}$ for any term $u \in \mathcal{T}(\Sigma, \mathcal{V})$. Furthermore, for $x \in X$, $[\![\sigma]\!]_{\mathfrak{M},\rho}(x) = [\![\sigma(x)]\!]_{\mathfrak{M},\rho} = \sigma(x)$ holds. Hence, by Lemma 5.3, $[\![\varphi]\!]_{\mathfrak{M},[\![\sigma]\!]_{\mathfrak{M},\rho}} = \mathsf{true}$. Thus, $[\![s]\!]_{\mathfrak{M},\rho} = [\![\ell]\!]_{\mathfrak{M},[\![\sigma]\!]_{\mathfrak{M},\rho}} = [\![r]\!]_{\mathfrak{M},[\![\sigma]\!]_{\mathfrak{M},\rho}} = [\![t]\!]_{\mathfrak{M},\rho}$. ◀

Now we present the soundness of our semantics with respect to the CE-validity.

▶ **Theorem 5.6** (soundness w.r.t. CE-validity). *Let $\mathfrak{T}$ be a CE-theory. If $\mathfrak{T} \models_{\mathsf{cec}} \Pi X.\, s \approx t \ [\varphi]$, then $\mathfrak{T} \models \Pi X.\, s \approx t \ [\varphi]$.*

Proof (Sketch). Let $\mathfrak{T} = \langle \mathcal{M}, \mathcal{E} \rangle$ and $\mathcal{M} = \langle \mathcal{I}, \mathcal{J} \rangle$. Suppose $\mathfrak{M} = \langle \mathfrak{I}, \mathfrak{J} \rangle$ is a CE-$\langle \Sigma, \mathcal{M} \rangle$-algebra such that $\models_{\mathfrak{M}} \mathcal{E}$. Let ρ be a valuation over $\mathfrak{M}$ satisfying the constraints φ and $\rho(x) \in \mathcal{I}(\tau)$ holds for all $x^\tau \in X$. Now, let $\hat{\rho}$ be a valuation that is obtained from ρ by restricting its domain to X. Then, $\models_{\mathcal{M}} \varphi\hat{\rho}$ by Lemma 5.3, and thus $s\hat{\rho} \overset{*}{\leftrightarrow}_{\mathcal{E}} t\hat{\rho}$ holds. Hence, by Lemma 5.5, $[\![s\hat{\rho}]\!]_{\mathfrak{M},\tau} = [\![t\hat{\rho}]\!]_{\mathfrak{M},\tau}$ holds for any valuation τ. This means that $[\![s]\!]_{\mathfrak{M},\tau'} = [\![t]\!]_{\mathfrak{M},\tau'}$ for any extension τ' of $\hat{\rho}$. In particular, one obtains $[\![s]\!]_{\mathfrak{M},\rho} = [\![t]\!]_{\mathfrak{M},\rho}$. ◀

The combination of Theorem 4.6 and Theorem 5.6 implies the following corollary.

▶ **Corollary 5.7** (soundness of $\mathbf{CEC}_0$ w.r.t. algebraic semantics). *Let $\mathfrak{T}$ be a CE-theory. If $\mathfrak{T} \vdash_{\mathbf{CEC}_0} \Pi X.\, s \approx t \ [\varphi]$, then $\mathfrak{T} \models \Pi X.\, s \approx t \ [\varphi]$.*

▶ **Example 5.8.** Consider integer arithmetic for the underlying model $\mathcal{M}$. Take a term signature $\mathcal{F}_{\mathsf{te}} = \{\mathsf{a} \colon \mathsf{Int}\}$. Consider the LCES $\mathcal{E} = \{\mathsf{a} \approx 0, \mathsf{a} \approx 1\}$ with $0, 1 \in \mathcal{V}\mathsf{al}$ and $0, 1 \in \mathbb{Z}$, hence $\mathcal{J}(0) = 0$ and $\mathcal{J}(1) = 1$. Then, for any valuation ρ on a CE-$\langle \Sigma, \mathcal{M} \rangle$-algebra $\mathfrak{M} = \langle \mathfrak{I}, \mathfrak{J} \rangle$ we have $\rho(0) = 0$ and $\rho(1) = 1$. Thus, if $\mathfrak{M}$ is a model of $\mathcal{E}$ then it follows that $0 = [\![0]\!] = [\![\mathsf{a}]\!] = [\![1]\!] = 1$, which is a contradiction. Therefore, there is no CE-$\langle \Sigma, \mathcal{M} \rangle$-algebra $\mathfrak{M}$ which validates $\mathcal{E}$.

This example motivates us to introduce the following definition of consistency for CE-theories.

▶ **Definition 5.9** (consistency). *A CE-theory is said to be* consistent *if it has a model.*

Our definition of consistent CE-theories does not exclude any theory that has only an almost trivial model such that $\mathcal{I}(\tau) = \{\bullet\}$ for all $\tau \in \mathcal{S}_{\mathsf{te}}$.

5.2 Completeness w.r.t. CE-Validity

In this subsection, we prove the completeness of algebraic semantics with respect to the CE-validity. That is, if a CE is valid in all models of a CE-theory then it is a CE-consequence of the CE-theory. We start by defining congruence relations, quotient algebras and term algebras that suit our formalism, incorporating standard notions for example the first-order equational logic, and then present basic results on them.

Let $\Sigma = \langle \mathcal{S}_{\mathsf{th}}, \mathcal{S}_{\mathsf{te}}, \mathcal{F}_{\mathsf{th}}, \mathcal{F}_{\mathsf{te}} \rangle$ be a signature, $\mathcal{M} = \langle \mathcal{I}, \mathcal{J} \rangle$ a model over $\mathcal{S}_{\mathsf{th}}$ and $\mathcal{F}_{\mathsf{th}}$, and $\mathfrak{M} = \langle \mathfrak{I}, \mathfrak{J} \rangle$ a CE-$\langle \Sigma, \mathcal{M} \rangle$-algebra. A *congruence relation* on $\mathfrak{M}$ is an $\mathcal{S}$-indexed family of relations $\sim = (\sim^\tau)_{\tau \in \mathcal{S}}$ that satisfies all of the following:

1. $\sim^\tau$ is an equivalence relation on $\mathfrak{I}(\tau)$,
2. $\sim^\tau \cap \mathcal{I}(\tau)^2$ is the identity relation for $\tau \in \mathcal{S}_{\mathsf{th}}$, and
3. for each $f \colon \tau_1 \times \cdots \times \tau_n \to \tau_0 \in \mathcal{F}$, if $a_i \sim^{\tau_i} b_i$ for all $1 \leqslant i \leqslant n$ then $\mathfrak{J}(f)(a_1, \ldots, a_n) \sim^{\tau_0} \mathfrak{J}(f)(b_1, \ldots, b_n)$.

We note here that the difference from the standard notion of congruence relation on algebras is located in item 2. Given a CE-$\langle \Sigma, \mathcal{M} \rangle$-algebra $\mathfrak{M} = \langle \mathfrak{I}, \mathfrak{J} \rangle$ and a congruence relation $\sim$ on it, the quotient CE-$\langle \Sigma, \mathcal{M} \rangle$-algebra $\mathfrak{M}/\!\sim = \langle \mathfrak{I}', \mathfrak{J}' \rangle$ is defined as follows: $\mathfrak{I}'(\tau) = \mathfrak{I}(\tau)/\!\sim^\tau = \{ [a]_{\sim^\tau} \mid a \in \mathfrak{I}(\tau) \}$ where $[a]_{\sim^\tau}$ is the equivalence class of $a \in \mathfrak{I}(\tau)$, i.e. $[a]_{\sim^\tau} = \{ b \in \mathfrak{I}(\tau) \mid a \sim^\tau b \}$, and $\mathfrak{J}'(f)([a_1]_{\sim^{\tau_1}}, \ldots, [a_n]_{\sim^{\tau_n}}) = [\mathfrak{J}(f)(a_1, \ldots, a_n)]_{\sim^{\tau_0}}$. It is easy to see that $\mathfrak{J}'$ is well-defined provided that $\sim$ is a congruence. When no confusion occurs, we omit the superscript τ from $\sim^\tau$.

▶ **Lemma 5.10** (quotient algebra). *Let $\mathfrak{M}$ be a CE-$\langle \Sigma, \mathcal{M} \rangle$-algebra, and $\sim$ a congruence on it. Then $\mathfrak{M}/\!\sim$ is a CE-$\langle \Sigma, \mathcal{M} \rangle$-algebra.*

Next, we define the term algebra. In contrast to the usual construction, for term CE-algebras we need to take care of identification induced by underlying models.

▶ **Definition 5.11** (term algebra). *Let $\Sigma = \langle \mathcal{S}_{\mathsf{th}}, \mathcal{S}_{\mathsf{te}}, \mathcal{F}_{\mathsf{th}}, \mathcal{F}_{\mathsf{te}} \rangle$ be a signature, $\mathcal{M} = \langle \mathcal{I}, \mathcal{J} \rangle$ a model over $\mathcal{S}_{\mathsf{th}}$ and $\mathcal{F}_{\mathsf{th}}$, and U a set of variables. The* term algebra generated from U with $\mathcal{M}$ *(denoted by $T[\mathcal{M}](\Sigma, U)$) is a pair $\mathfrak{M} = \langle \mathfrak{I}, \mathfrak{J} \rangle$ where*

- $\mathfrak{I}(\tau) = \mathcal{T}(\mathcal{F}, U)^\tau / \!\sim_{\mathrm{c}}$, *and*
- $\mathfrak{J}(f)([t_1]_{\mathrm{c}}, \ldots, [t_n]_{\mathrm{c}}) = [f(t_1, \ldots, t_n)]_{\mathrm{c}}$ *for any $f \in \mathcal{F}$.*

Here, $\mathcal{F} = \mathcal{F}_{\mathsf{th}} \cup \mathcal{F}_{\mathsf{te}}$, $\sim_{\mathrm{c}} = \leftrightarrow^*_{\mathsf{calc}}$, and $[t]_{\mathrm{c}}$ denotes the $\sim_{\mathrm{c}}$-equivalence class containing a term t. Since $\leftrightarrow^*_{\mathsf{calc}}$ is sort preserving, we regard $\sim_{\mathrm{c}}$ as the sum of the τ-indexed family of relations $\sim^\tau_{\mathrm{c}}$ with $\tau \in \mathcal{S}$. Clearly, $\mathfrak{J}(f)$ is well-defined, since $\leftrightarrow^*_{\mathsf{calc}}$ is closed under contexts.

▶ **Lemma 5.12.** *Let $\Sigma = \langle \mathcal{S}_{\mathsf{th}}, \mathcal{S}_{\mathsf{te}}, \mathcal{F}_{\mathsf{th}}, \mathcal{F}_{\mathsf{te}} \rangle$ be a signature, $\mathcal{M}$ a model over $\mathcal{S}_{\mathsf{th}}$ and $\mathcal{F}_{\mathsf{th}}$, and U a set of variables. Then, the term algebra $T[\mathcal{M}](\Sigma, U)$ is a CE-$\langle \Sigma, \mathcal{M} \rangle$-algebra.*

We introduce a syntactic counter part of the notion of consistency of CE-theories for which equivalence of the two notions will be proved only briefly.

▶ **Definition 5.13** (consistency w.r.t. values)**.** *A CE-theory $\mathfrak{T} = \langle \mathcal{M}, \mathcal{E} \rangle$ is said to be* consistent with respect to values *(value-consistent, for short) if for any $u, v \in \mathcal{V}\mathsf{al}^\tau$, $u \overset{*}{\leftrightarrow}_{\mathcal{E}} v$ implies $u = v$.*

Based on the preparations so far, we now proceed to construct canonical models of CE-theories. The first step is to show that $\overset{*}{\leftrightarrow}_{\mathcal{E}}$ is a congruence relation on the term algebra for each CE-theory $\mathfrak{T} = \langle \mathcal{M}, \mathcal{E} \rangle$; special attention on $\sim_{\mathrm{c}}$ is required.

▶ **Lemma 5.14.** *Let $\mathfrak{T} = \langle \mathcal{M}, \mathcal{E} \rangle$ be a value-consistent CE-theory over a signature Σ, and U a set of variables. For any $[s]_{\mathrm{c}}, [t]_{\mathrm{c}} \in T[\mathcal{M}](\Sigma, U)$, let $\sim_{\mathcal{E}} = \{ \langle [s]_{\mathrm{c}}, [t]_{\mathrm{c}} \rangle \mid s \overset{*}{\leftrightarrow}_{\mathcal{E}} t \}$. Then, $\sim_{\mathcal{E}}$ is a congruence relation on the term algebra $T[\mathcal{M}](\Sigma, U)$.*

Proof (Sketch). Note first that $\sim_{\mathcal{E}}$ is well-defined because one has always $\leftrightarrow^*_{\mathsf{calc}} \subseteq \overset{*}{\leftrightarrow}_{\mathcal{E}}$. Let $\Sigma = \langle \mathcal{S}_{\mathsf{th}}, \mathcal{S}_{\mathsf{te}}, \mathcal{F}_{\mathsf{th}}, \mathcal{F}_{\mathsf{te}} \rangle$, $\mathcal{M} = \langle \mathcal{I}, \mathcal{J} \rangle$, and $T[\mathcal{M}](\Sigma, U) = \langle \mathfrak{I}, \mathfrak{J} \rangle$. We only present a proof that $\sim_{\mathcal{E}}^{\tau} \cap \mathcal{I}(\tau)^2$ equals the identity relation for $\tau \in \mathcal{S}_{\mathsf{th}}$ here. Let $\tau \in \mathcal{S}_{\mathsf{th}}$ and suppose $[u]_{\mathrm{c}} \sim_{\mathcal{E}}^{\tau} [v]_{\mathrm{c}}$ with $u, v \in \mathcal{I}(\tau) \cong \mathcal{V}\mathsf{al}^\tau$. Then, we have $u \overset{*}{\leftrightarrow}_{\mathcal{E}} v$ by the definition of $\sim_{\mathcal{E}}$, and by consistency w.r.t. values of the theory $\mathfrak{T}$, we obtain $u = v$ as $u, v \in \mathcal{V}\mathsf{al}$. Therefore, $[u]_{\mathrm{c}} = [v]_{\mathrm{c}}$. ◀

We give a construction of canonical models for each CE-theory $\mathcal{T}$.

▶ **Lemma 5.15.** *Let $\mathfrak{T} = \langle \mathcal{M}, \mathcal{E} \rangle$ be a value-consistent CE-theory over a signature Σ. Then, the quotient $\mathcal{T}_{\mathcal{E}} = T[\mathcal{M}](\Sigma, \mathcal{V})/\sim_{\mathcal{E}}$ of the term algebra is a CE-$\langle \Sigma, \mathcal{M} \rangle$-algebra. Furthermore, both of the following hold:*

1. *$\models_{\mathcal{T}_{\mathcal{E}}} \Pi X.\, s \approx t \ [\varphi]$ if and only if $\mathcal{E} \models_{\mathsf{cec}} \Pi X.\, s \approx t \ [\varphi]$, and*
2. *$\models_{\mathcal{T}_{\mathcal{E}}} \mathcal{E}$.*

Proof (Sketch). That $\mathcal{T}_{\mathcal{E}}$ is a CE-$\langle \Sigma, \mathcal{M} \rangle$-algebra follows from Lemmas 5.10 and 5.14. Let us abbreviate $[[t]_{\mathrm{c}}]_{\sim_{\mathcal{E}}}$ as $[t]_{\mathcal{E}}$. First we claim that $[u\sigma]_{\mathcal{E}} = [\![u]\!]_{\mathcal{T}_{\mathcal{E}}, \rho}$ holds for any term u, for any substitution σ and valuation ρ on $\mathcal{T}_{\mathcal{E}}$ such that $\rho(x) = [\sigma(x)]_{\mathcal{E}}$, using induction on u.

1. ($\Rightarrow$) Let σ be an X-valued substitution such that $\models_{\mathcal{M}} \varphi\sigma$. Take a valuation ρ on $\mathcal{T}_{\mathcal{E}}$ as $\rho(x) = [\sigma(x)]_{\mathcal{E}}$. Then, $\rho(x) \in \mathcal{I}(\tau)$ for all $x^\tau \in X$ and $\models_{\mathcal{M}} \varphi\rho$. Thus, $[\![s]\!]_{\mathcal{T}_{\mathcal{E}}, \rho} = [\![t]\!]_{\mathcal{T}_{\mathcal{E}}, \rho}$. Hence, $[s\sigma]_{\mathcal{E}} = [t\sigma]_{\mathcal{E}}$ by the claim, and therefore, $s\sigma \overset{*}{\leftrightarrow}_{\mathcal{E}} t\sigma$. ($\Leftarrow$) Let ρ be a valuation over $\mathcal{T}_{\mathcal{E}}$ satisfying the constraints φ and $\rho(x) \in \mathcal{I}(\tau)$ for all $x \in X$. Take a substitution σ in such a way that $\sigma(x) = v_x$ for each $x \in X$, where $v_x \in \mathcal{V}\mathsf{al}^\tau$ such that $[v_x]_{\mathcal{E}} = \rho(x)$. By Lemma 5.3, $[\![\varphi]\!]_{\mathcal{M}, \rho} = \mathsf{true}$, and thus, $\models_{\mathcal{M}} \varphi\sigma$ by Lemma 2.1. Hence $s\sigma \overset{*}{\leftrightarrow}_{\mathcal{E}} t\sigma$, and thus $[s\sigma]_{\mathcal{E}} = [t\sigma]_{\mathcal{E}}$. Therefore $[\![s]\!]_{\mathcal{T}_{\mathcal{E}}, \rho} = [\![t]\!]_{\mathcal{T}_{\mathcal{E}}, \rho}$ by the claim.

Item **2** follows from item **1**. ◀

Before proceeding to the completeness theorem, we connect the two notions related to consistency (Definitions 5.9 and 5.13).

▶ **Lemma 5.16.** *A CE-theory $\mathfrak{T}$ is consistent if and only if it is consistent with respect to values.*

Proof (Sketch). ($\Rightarrow$) Suppose that $\mathfrak{T} = \langle \mathcal{M}, \mathcal{E} \rangle$ is a CE-theory over a signature Σ and let $\mathcal{M} = \langle \mathcal{I}, \mathcal{J} \rangle$. Let $\mathfrak{M} = \langle \mathfrak{I}, \mathfrak{J} \rangle$ be a CE-$\langle \Sigma, \mathcal{M} \rangle$-algebra such that $\models_{\mathfrak{M}} \mathcal{E}$. Suppose $u, v \in \mathcal{V}\mathsf{al}^\tau$ with $u \overset{*}{\leftrightarrow}_{\mathcal{E}} v$. Then $[\![u]\!]_{\mathfrak{M}} = [\![v]\!]_{\mathfrak{M}}$ by Lemma 5.5. Therefore, by $u, v \in \mathcal{V}\mathsf{al}^\tau \cong \mathcal{I}(\tau) \subseteq \mathfrak{I}(\tau)$, we have $u = [\![u]\!]_{\mathcal{M}} = [\![u]\!]_{\mathfrak{M}} = [\![v]\!]_{\mathfrak{M}} = [\![v]\!]_{\mathcal{M}} = v$. ($\Leftarrow$) By Lemma 5.15, $\mathcal{T}_{\mathcal{E}}$ is a model of $\mathfrak{T}$. This witnesses that $\mathfrak{T}$ is consistent. ◀

We now arrive at the main theorem of this section.

▶ **Theorem 5.17** (completeness). *Let $\mathfrak{T} = \langle \mathcal{M}, \mathcal{E} \rangle$ be a consistent CE-theory. Then, we have $\mathcal{E} \models_{\mathsf{cec}} \Pi X.\, s \approx t\ [\varphi]$ if and only if $\mathcal{E} \models \Pi X.\, s \approx t\ [\varphi]$.*

Proof. The *only if* part follows from Theorem 5.6. Thus, it remains to show the *if* part. Suppose contrapositively that $\mathcal{E} \models_{\mathsf{cec}} \Pi X.\, s \approx t\ [\varphi]$ does not hold. Then, by Lemma 5.15 **1**, $\not\models_{\mathcal{T}_\mathcal{E}} \Pi X.\, s \approx t\ [\varphi]$. Since $\models_{\mathcal{T}_\mathcal{E}} \mathcal{E}$, by Lemma 5.15 **2**, this witnesses that there exists a CE-$\langle \Sigma, \mathcal{M} \rangle$-algebra $\mathfrak{M}$ such that $\models_{\mathfrak{M}} \mathcal{E}$ but not $\models_{\mathfrak{M}} \Pi X.\, s \approx t\ [\varphi]$. This means $\mathcal{E} \not\models \Pi X.\, s \approx t\ [\varphi]$. This completes the proof of the *if* part. ◀

To conclude this section, we explain the postponed question from the beginning of the section on the definition of CE-algebras. The question was on why it is required to include those equipped with underlying extended models – if such models would not be allowed, one does not obtain the completeness result as witnessed by the following example.

▶ **Example 5.18.** Consider integer arithmetic for the underlying model $\mathcal{M}$. Take a term signature $\mathcal{F}_{\mathsf{te}} = \{\mathsf{f} \colon \mathsf{Ints} \to \mathsf{Bool}, \mathsf{g} \colon \mathsf{Ints} \to \mathsf{Bool}\}$. Consider the LCES $\mathcal{E} = \{\mathsf{f}(x) \approx \mathsf{true}\ [x \geqslant 0], \mathsf{f}(x) \approx \mathsf{true}\ [x < 0], \mathsf{g}(x) \approx \mathsf{true}\}$. By orienting the equations in an obvious way, we obtain a complete LCTRS (e.g. [13, 20]). Then as $\mathsf{g}(x)\!\downarrow = \mathsf{true} \neq \mathsf{f}(x) = \mathsf{f}(x)\!\downarrow$, it turns out that no conversions hold between $\mathsf{g}(x)$ and $\mathsf{f}(x)$. It follows from the Theorem 3.17 that $\mathcal{E} \not\models_{\mathsf{cec}} \Pi\varnothing.\, \mathsf{g}(x) \approx \mathsf{f}(x)$. Now, from Theorem 5.17, we have $\mathcal{E} \not\models \Pi\varnothing.\, \mathsf{g}(x) \approx \mathsf{f}(x)$, i.e. one should find a model that witnesses this invalidity. Indeed, one can take a CE-$\langle \Sigma, \mathcal{M} \rangle$-algebra $\mathfrak{M} = \langle \mathfrak{I}, \mathfrak{J} \rangle$ with $\mathfrak{I}(\mathsf{Ints}) = \mathbb{Z} \cup \{\bullet\}$, where $\bullet \notin \mathbb{Z}$, with the interpretations: $\mathfrak{J}(\mathsf{f})(\bullet) = \mathsf{false}$, $\mathfrak{J}(\mathsf{f})(a) = \mathsf{true}$ for all $a \in \mathbb{Z}$, and $\mathfrak{J}(\mathsf{g})(x) = \mathsf{true}$ for all $x \in \mathbb{Z} \cup \{\bullet\}$. On the other hand, if we would require to take $\mathfrak{I}(\mathsf{Ints}) = \mathbb{Z}$, then we do not get any model that invalidates this CE.

6 Related Work

Constrained rewriting began to be popular around 1990, which has been initiated by the motivation to achieve a tractable solution for completion modulo equations (such as AC, ACI, etc.), by separating off the (intractable) equational solving part as constraints. These constraints mainly consist of (dis)equality of built-in equational theories such as $x{*}y \approx_{AC} y{*}x$. A constrained completion procedure in such a framework is given in [10]; it is well-known that the specification language Maude also deals with such built-in theories [14]. This line of research was extended to a framework of rewriting with constraints of an arbitrary first-order formula in [10], where various completion methods have been developed for this. However, they, similar to us, mainly considered term algebras as the underlying models, because the main motivation was to deal with a wide range of completion problems by separating off some parts of the equational theory as constraints.

Another well-known type of constraints studied in the context of constrained rewriting is membership constraints of regular tree languages. This type of constraints is motivated by dealing with terms over an (order-)sorted signature and representing an infinite number of terms that obeys regular patterns obtained from divergence of theorem proving procedures. In this line of research, [5, 6] give a dedicated completion method for constrained rewrite systems with membership constraints of regular tree languages. Further a method for inductive theorem proving for conditional and constrained systems, which is based on tree grammars with constraints, has been proposed in [3]. We also want to mention [18] as a formalism with more abstract constraints – confluence of term rewrite systems with membership constraints over an arbitrary term set has been considered there.

The work in this era which is in our opinion closest to the LCTRSs formalism is the one given in [7]. This is also motivated by giving a link between (symbolic) equational deduction and constraint solving. Thus, they considered constraints of an arbitrary theory such as linear integer arithmetic, similarly to LCTRSs. Based on the initial model of this framework, they gave an operational semantics of constraint equational logic programming.

The introduction of the LCTRS framework is more recent, and was initiated by the motivation to deal with built-in data structures such as integers, bit-vectors etc. in order to verify programs written by real-world programming languages with the help of SMT-solvers. A detailed comparison to the works in this line of research has been given in [13].

All in all, to the best of our knowledge, there does not exist anything in the literature on algebraic semantics of constrained rewriting and Birkhoff style completeness, as considered in this paper.

7 Conclusion

With the goal to establish a semantic formalism of logically constrained rewriting, we have introduced the notions of constrained equations and CE-theories. For this, we have extended the form of these constrained equations by specifying explicitly the variables, which need to be instantiated by values, in order to treat equational properties in a uniform way. Then we have introduced a notion of CE-validity to give a uniform foundation from a semantic point of view for the LCTRS formalism. After establishing basic properties of the introduced validity, we have shown the relation to the conversion of rewriting. Then we presented a sound inference system $\mathbf{CEC}_0$ to prove validity of constrained equations in CE-theories. We have demonstrated its ability to establish validity via some examples. A partial completeness result and a discussion on why only partial completeness is obtained followed. Finally, we devised sound and complete algebraic semantics, which enables one to show invalidity of constrained equations in CE-theories. Furthermore, we have derived an important characterization of CE-theories, namely, consistency of CE-theories, for which the completeness theorem holds. Thus, we have established a basis for CE-theories and their validity by showing its fundamental properties and giving methods for proving and disproving the validity of constrained equations in CE-theories.

The question whether there exists a sound and complete proof system that captures CE-validity remains open. Part of our future work is the automation of proving validity of constrained equations.

This paper uses the initial formalism of LCTRSs given in [13]. However, there exists a variant which incorporates the interpretation of user-defined function symbols by the term algebra [4]. This variant is incomparable to the initial one. Nevertheless, to investigate the semantic side of LCTRSs, the initial formalism was a reasonable starting point. The adaptation of the current work to the extended formalism is also a part of our future work.

References

1 Takahito Aoto, Naoki Nishida, and Jonas Schöpf. Equational theories and validity for logically constrained term rewriting (full version). *CoRR*, abs/2405.01174, 2024. doi:10.48550/arXiv.2405.01174.

2 Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998. doi:10.1145/505863.505888.

3 Adel Bouhoula and Florent Jacquemard. Sufficient completeness verification for conditional and constrained TRS. *Journal of Applied Logic*, 10(1):127–143, 2012. doi:10.1016/J.JAL.2011.09.001.

4 Ştefan Ciobâcă and Dorel Lucanu. A coinductive approach to proving reachability properties in logically constrained term rewriting systems. In *Proceedings of the 9th IJCAR*, volume 10900 of *Lecture Notes in Computer Science*, pages 295–311. Springer, 2018. doi:10.1007/978-3-319-94205-6_20.

5 Hubert Comon. Completion of rewrite systems with membership constraints. part I: deduction rules. *Journal of Symbolic Computation*, 25(4):397–419, 1998. doi:10.1006/JSCO.1997.0185.

6 Hubert Comon. Completion of rewrite systems with membership constraints. part II: constraint solving. *Journal of Symbolic Computation*, 25(4):421–453, 1998. doi:10.1006/JSCO.1997.0186.

7 John Darlington and Yike Guo. Constrained equational deduction. In *Proceedings of the 2nd CTRS*, volume 516 of *Lecture Notes in Computer Science*, pages 424–435. Springer, 1991. doi:10.1007/3-540-54317-1_111.

8 Heinz-Dieter Ebbinghaus, Jörg Flum, Wolfgang Thomas, and Ann S Ferebee. *Mathematical logic*, volume 1910. Springer, 1994.

9 Carsten Fuhs, Cynthia Kop, and Naoki Nishida. Verifying procedural programs via constrained rewriting induction. *ACM Transactions on Computational Logic*, 18(2):14:1–14:50, 2017. doi:10.1145/3060143.

10 Claude Kirchner and Hélène. Kirchner. Constrained equational reasoning. In *Proceedings of the 14th ISSAC*, pages 3824–389. ACM, 1989. doi:10.1145/74540.74585.

11 Misaki Kojima and Naoki Nishida. From starvation freedom to all-path reachability problems in constrained rewriting. In *Proceedings of the 25th PADL*, volume 13880 of *Lecture Notes in Computer Science*, pages 161–179. Springer Nature Switzerland, 2023. doi:10.1007/978-3-031-24841-2_11.

12 Misaki Kojima and Naoki Nishida. Reducing non-occurrence of specified runtime errors to all-path reachability problems of constrained rewriting. *Journal of Logical and Algebraic Methods in Programming*, 135:1–19, 2023. doi:10.1016/j.jlamp.2023.100903.

13 Cynthia Kop and Naoki Nishida. Term rewriting with logical constraints. In *Proceedings of the 9th FroCoS*, volume 8152 of *Lecture Notes in Computer Science*, pages 343–358, 2013. doi:10.1007/978-3-642-40885-4_24.

14 José Meseguer. Twenty years of rewriting logic. *Journal of Logic and Algebraic Programming*, 81(7-8):721–781, 2012. doi:10.1016/J.JLAP.2012.06.003.

15 Naoki Nishida and Sarah Winkler. Loop detection by logically constrained term rewriting. In *Proceedings of the 10th VSTTE*, volume 11294 of *Lecture Notes in Computer Science*, pages 309–321. Springer, 2018. doi:10.1007/978-3-030-03592-1_18.

16 Enno Ohlebusch. *Advanced Topics in Term Rewriting*. Springer, 2002. doi:10.1007/978-1-4757-3661-8.

17 Jonas Schöpf and Aart Middeldorp. Confluence criteria for logically constrained rewrite systems. In *Proceedings of the 29th CADE*, volume 14132 of *Lecture Notes in Artificial Intelligence*, pages 474–490, 2023. doi:10.1007/978-3-031-38499-8_27.

18 Yoshihito Toyama. Membership conditional term rewriting systems. *IEICE Transactions*, E72(11):1224–1229, 1989.

19 Dirk van Dalen. *Logic and Structure*. Springer-Verlag, Berlin, third edition, 1994.

20 Sarah Winkler and Aart Middeldorp. Completion for logically constrained rewriting. In *Proceedings of the 3rd FSCD*, volume 108 of *LIPIcs*, pages 30:1–30:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.FSCD.2018.30.

21 Sarah Winkler and Georg Moser. Runtime complexity analysis of logically constrained rewriting. In *Proceedings of the 30th LOPSTR*, volume 12561 of *Lecture Notes in Computer Science*, pages 37–55, 2021. doi:10.1007/978-3-030-68446-4_2.

Termination of Generalized Term Rewriting Systems

Salvador Lucas
DSIC & VRAIN, Universitat Politècnica de València, Spain, Spain

Abstract

We investigate termination of *Generalized Term Rewriting Systems* (GTRSs), which extend *Conditional Term Rewriting Systems* by considering replacement restrictions on selected arguments of function symbols, as in *Context-Sensitive Rewriting*, and conditional rewriting rules whose conditional part may include not only a mix of the usual (reachability, joinability,...) conditions, but also atoms defined by a set of definite Horn clauses. GTRSs can be used to prove confluence and termination of *Generalized Rewrite Theories* and Maude programs. We have characterized confluence of terminating GTRSs as the joinability of a finite set of conditional pairs. Since termination of GTRSs is underexplored to date, this paper introduces a Dependency Pair Framework which is well-suited to automatically (dis)prove termination of GTRSs.

2012 ACM Subject Classification Theory of computation → Automated reasoning; Theory of computation → Logic and verification; Theory of computation → Equational logic and rewriting

Keywords and phrases Program Analysis, Reduction-Based Systems, Termination

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.32

Funding Supported by MCIN/AEI/10.13039/501100011033 and by "ERDF A way of making Europe" (PID2021-122830OB-C42) and by the grant CIPROM/2022/6 funded by Generalitat Valenciana.

Acknowledgements I thank the reviewers for their useful comments leading to several improvements in the paper. I also thank Raúl Gutiérrez for his comments.

1 Introduction

Generalized Term Rewriting Systems (GTRS [21]) extend Conditional Term Rewriting Systems (CTRSs, see, e.g., [29, Chapter 7] and the references therein) by (i) restricting reductions on specific arguments of function symbols by means of a replacement map μ [17] and by also (ii) including atoms in the conditional part of rules which are defined by (iii) a set of definite Horn clauses. GTRSs and CTRSs are compared in [21, Section 7.3].

▶ **Example 1.** The following GTRS $\mathcal{R}$ to divide natural numbers in Peano's notation (adapted from [32, Example 9]), consists of clauses (1)-(5) and rules (6)-(10).

$$x \approx y \Leftarrow x \to^* y \quad (1) \qquad x - 0 \to x \quad (6)$$
$$s(x) > 0 \quad (2) \qquad 0 - y \to 0 \quad (7)$$
$$s(x) > s(y) \Leftarrow x > y \quad (3) \qquad \mathsf{s}(x) - \mathsf{s}(y) \to x - y \quad (8)$$
$$0 \le x \quad (4) \qquad \mathsf{div}(x, y) \to \mathsf{pair}(0, x) \Leftarrow y > x \quad (9)$$
$$s(x) \le s(y) \Leftarrow x \le y \quad (5)$$
$$\mathsf{div}(x, y) \to \mathsf{pair}(\mathsf{s}(q), r) \Leftarrow y \le x, \mathsf{div}(x - y, y) \approx \mathsf{pair}(q, r) \quad (10)$$

A call $\mathsf{div}(m, n)$ would return $\mathsf{pair}(q, r)$ with q and r the quotient and remainder.

Rewriting steps $s \to_{\mathcal{R}} t$ with GTRSs $\mathcal{R}$ are defined by *deduction* of goals $s \to t$ (where $\to$ is a *predicate symbol*) with respect to the first-order theory $\overline{\mathcal{R}}$ of $\mathcal{R}$, i.e., $s \to_{\mathcal{R}} t$ iff $\overline{\mathcal{R}} \vdash s \to t$ holds. Accordingly, main computational properties such as *confluence* and *termination* are

defined by applying the usual abstract notions to $\to_{\mathcal{R}}$. For instance, a GTRS $\mathcal{R}$ is *terminating* if $\to_{\mathcal{R}}$ is terminating, i.e., there is no infinite rewrite sequence $t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} \cdots$. Confluence of terminating GTRSs $\mathcal{R}$ is characterized as the joinability of a number of conditional pairs $\langle s, t\rangle \Leftarrow c$ obtained from the rules in $\mathcal{R}$ [21]. In this paper we investigate how to prove *termination of GTRSs*. As explained in [24], the "termination behavior" (of CTRSs) can be investigated as having a *horizontal dimension* (H-termination, or just termination in the usual sense), which pays attention to *sequences* of rewriting steps only, and a *vertical dimension* (V-termination) which pays attention to the existence of *infinite proof trees*, eventually built to prove a rewriting goal $s \to t$ in a proof system as in natural deduction [30]. When *both* H- and V-termination are achieved, the CTRS is said to be *operationally terminating* [22]. Thus, termination is a weaker property, *easier* to achieve. For instance, $\mathcal{R}$ in Example 1 is *not* operationally terminating: the attempt to rewrite $\mathsf{div}(0,0)$ using rule (10) leads to build an infinite tree due to the (recurrent) need to prove that $\mathsf{div}(0,0) \approx \mathsf{pair}(q,r)$ (obtained from the second condition of the rule after simplifying $\mathsf{div}(0-0,0) \approx \mathsf{pair}(q,r)$) is satisfied by further rewriting on $\mathsf{div}(0,0)$. However, $\mathcal{R}$ is *terminating* (see Example 43).

The paper is organized as follows: after some preliminaries in Section 2, Section 3 recalls the notion of a GTRS. Section 4 investigates the structure of infinite rewrite sequences with GTRSs. Nowadays, proofs of termination of reduction-based systems are based on the notion of *dependency pair* (DP [2]) and *dependency pair framework* [8, 9] for Term Rewriting Systems (TRSs [3]). Section 5 introduces appropriate notions of dependency pairs of a GTRS $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H, R)$, which are viewed as a new set P of Horn clauses which are added to H to obtain a new GTRS $\mathsf{DP}_{HC}(\mathcal{R})$ which characterizes termination of $\mathcal{R}$. Section 6 introduces a framework for (dis)proving termination of GTRSs which is amenable for automation through appropriate adaptations of the usual notions of *termination problem* and *processor* introduced in the DP Framework for TRSs [8]. Section 7 introduces five processors to be used in our framework and illustrates their use by means of examples. Section 8 discusses some related work. Section 9 concludes.

2 Preliminaries

In the following, we often write *iff* instead of *if and only if*. We assume some familiarity with the basic notions of term rewriting [3, 29, 33] and first-order logic [7, 27]. We just summarize the main notions and notations we use.

Given a binary relation $\mathsf{R} \subseteq A \times A$ on a set A, we often write $a \ \mathsf{R} \ b$ instead of $(a,b) \in \mathsf{R}$. The *transitive* closure of R is denoted by R^+, and its *reflexive and transitive* closure by R^*. An element $a \in A$ is *irreducible* (or an R-*normal form*), if there exists no b such that $a \ \mathsf{R} \ b$. We say that R is *terminating* if there is no infinite sequence $a_1 \ R \ a_2 \ R \ a_3 \cdots$. In this paper, $\mathcal{X}$ denotes a countable set of *variables* and $\mathcal{F}$ denotes a *signature*, i.e., a set of *symbols* $\{f, g, \ldots\}$, each with a fixed *arity* given by a mapping $ar : \mathcal{F} \to \mathbb{N}$. The set of terms built from $\mathcal{F}$ and $\mathcal{X}$ is $\mathcal{T}(\mathcal{F}, \mathcal{X})$. The set of variables occurring in t is $\mathcal{V}ar(t)$. Terms are viewed as labeled trees in the usual way. *Positions* p are represented by chains of positive natural numbers used to address subterms $t|_p$ of t. The root position of a term is denoted as Λ; the root symbol as $root(t)$. The *set of positions* of a term t is $\mathcal{P}os(t)$. A term t is a *strict* subterm of s (written $s \rhd t$) iff $t = s|_p$ for some $p \in \mathcal{P}os(s) - \{\Lambda\}$. We write $s \trianglerighteq t$ if $s = t$ or $s \rhd t$.

Given a signature $\mathcal{F}$, a *replacement map* is a mapping μ from symbols in $\mathcal{F}$ to sets of positive numbers satisfying $\mu(f) \subseteq \{1, \ldots, ar(f)\}$ for all $f \in \mathcal{F}$ [17]. The set of replacement maps for $\mathcal{F}$ is $M_{\mathcal{F}}$. We use $\mu_\top(f) = \{1, \ldots, ar(f)\}$ and $\mu_\bot(f) = \emptyset$ for all $f \in \mathcal{F}$. The set of μ-

replacing (or active*) positions* of t is $\mathcal{P}os^{\mu}(t) = \{\Lambda\}$, if $t \in \mathcal{X}$, and $\mathcal{P}os^{\mu}(t) = \{\Lambda\} \cup \{i.p \mid i \in \mu(f), p \in \mathcal{P}os^{\mu}(t_i)\}$, if $t = f(t_1, \ldots, t_k)$. The set of *non-μ-replacing* (or *frozen*) positions of t is $\overline{\mathcal{P}os^{\mu}}(t) = \mathcal{P}os(t) - \mathcal{P}os^{\mu}(t)$. Accordingly, subterms $u = t|_p$ of a term t at an active (resp. frozen) position $p \in \mathcal{P}os^{\mu}(t)$ (resp. $p \in \overline{\mathcal{P}os^{\mu}}(t)$) of t are called *active* (resp. *frozen*). Positions of *active* non-variable subterms of t are denoted as $\mathcal{P}os^{\mu}_{\mathcal{F}}(t)$. Given a term t, the set of variables occurring at active positions in t is $\mathcal{V}ar^{\mu}(t) = \{x \in \mathcal{V}ar(t) \mid \exists p \in \mathcal{P}os^{\mu}(t), x = t|_p\}$. A term t is a *strict, active* subterm of s (written $s \rhd_{\mu} t$) iff $t = s|_p$ for some $p \in \mathcal{P}os^{\mu}(s) - \{\Lambda\}$. We write $s \trianglerighteq_{\mu} t$ if $s = t$ or $s \rhd_{\mu} t$.

Consider a signature $\mathcal{F}$ of *function symbols* and a signature Π of *predicate symbols*. Atoms $A \in Atoms(\mathcal{F}, \Pi, \mathcal{X})$ and first-order formulas $F \in Forms(\mathcal{F}, \Pi, \mathcal{X})$ on such signatures, with variables in $\mathcal{X}$, are built in the usual way. A (definite) Horn clause (with label α) is written $\alpha : A \Leftarrow A_1, \ldots, A_n$, for atoms $A, A_1, \ldots, A_n$; if $n = 0$, then α is written A rather than $A \Leftarrow$. A first-order theory (FO-theory for short) Th is a set of sentences (formulas whose variables are all *quantified*). An $\mathcal{F}, \Pi$-*structure* $\mathcal{A}$ (or just *structure* if no confusion arises) consists of a *non-empty* set $\mathsf{dom}(\mathcal{A})$, called *domain* and often denoted $\mathcal{A}$ if no confusion arises, together with an interpretation of symbols $f \in \mathcal{F}$ and $P \in \Pi$ as mappings $f^{\mathcal{A}}$ and relations $P^{\mathcal{A}}$ on $\mathcal{A}$, respectively. Then, the usual interpretation of first-order formulas with respect to $\mathcal{A}$ is considered [27, page 60]. An $\mathcal{F}, \Pi$-model for a theory Th is just a structure $\mathcal{A}$ that makes them all true, written $\mathcal{A} \models \mathsf{Th}$. A formula F is a *logical consequence* of a theory Th (written $\mathsf{Th} \models F$) iff every model $\mathcal{A}$ of Th is also a model of F. Also, $\mathsf{Th} \vdash F$ means that F is *deducible* from Th by using a correct and complete deduction procedure. In that case, $\vdash$ and $\models$ coincide.

An *f-condition* γ is an atom [13]. Sequences $\mathsf{F} = (\gamma_i)_{i=1}^{n} = (\gamma_1, \ldots, \gamma_n)$ of f-conditions are called *f-sequences*. We often drop "f-" when no confusion arises. Empty sequences are written $()$. Given an FO-theory Th, a condition γ is Th-*feasible* (or just *feasible* if no confusion arises) if $\mathsf{Th} \vdash \sigma(\gamma)$ holds for some substitution σ; otherwise, it is *infeasible*. Note that (in)feasibility is, in general, undecidable. A sequence F is Th-feasible (or just *feasible*) iff there is a substitution σ such that, for all $\gamma \in \mathsf{F}$, $\mathsf{Th} \vdash \sigma(\gamma)$ holds.

A CTRS is a pair $\mathcal{R} = (\mathcal{F}, R)$ where $\mathcal{F}$ is a signature and R is a set of rules $\ell \to r \Leftarrow c$, with c a sequence $s_1 \approx t_1, \cdots, s_n \approx t_n$ for some $n \geq 0$ and terms $\ell, r, s_1, \ldots, t_n$ such that $\ell \notin \mathcal{X}$. As usual, ℓ and r are called the *left-* and *right-hand sides* of the rule (*lhs* and *rhs*, respectively), and c is the *conditional part* of the rule. *Labeled* rules are written $\alpha : \ell \to r \Leftarrow c$, where α is a *label*. In the following, given $\mathcal{R}$, we often write $\alpha \in \mathcal{R}$, instead of $\alpha \in R$, to say that α is a rule of $\mathcal{R}$.

3 Generalized Term Rewriting Systems

A *Generalized Term Rewriting System (GTRS)* is a tuple $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H, R)$ where $\mathcal{F}$ is a signature of *function symbols*, Π is a signature of *predicate symbols*, including at least $\to$ and $\to^*$, $\mu \in M_{\mathcal{F}}$ ($\mu_{\top}$ is assumed if μ is not explicitly given, as in Example 1), H is a (possibly empty) set of clauses $A \Leftarrow c$, where $root(A) \notin \{\to, \to^*\}$, and R is a set of rewrite rules $\ell \to r \Leftarrow c$ such that $\ell \notin \mathcal{X}$. In both cases, c is a sequence of atoms. Note that rules in R are Horn clauses. We often give them a label α as follows: $\alpha : \ell \to r \Leftarrow c$. The rules in R permit the usual distinction of function symbols $f \in \mathcal{F}$ as *defined* (if $root(\ell) = f$ for some $\ell \to r \Leftarrow c \in R$) or *constructor* symbols (otherwise). The set of defined (resp. constructor) symbols of $\mathcal{R}$ is $\mathcal{D}_{\mathcal{R}}$ or just $\mathcal{D}$ if no confusion arises (resp. $\mathcal{C}_{\mathcal{R}}$ or $\mathcal{C}$). The FO-theory of $\mathcal{R}$ is

$$\overline{\mathcal{R}} = \{(\mathrm{Rf}), (\mathrm{Co})\} \cup \{(\mathrm{Pr})_{f,i} \mid f \in \mathcal{F}, i \in \mu(f)\} \cup \{(\mathrm{HC})_{\alpha} \mid \alpha \in H \cup R\}$$

Table 1 Generic sentences of the first-order theory of rewriting.

Label	Sentence
(Rf)	$(\forall x)\ x \to^* x$
(Co)	$(\forall x, y, z)\ x \to y \wedge y \to^* z \Rightarrow x \to^* z$
$(\mathrm{Pr})_{f,i}$	$(\forall x_1, \ldots, x_k, y_i)\ x_i \to y_i \Rightarrow f(x_1, \ldots, x_i, \ldots, x_k) \to f(x_1, \ldots, y_i, \ldots, x_k)$
$(\mathrm{HC})_{A \Leftarrow A_1, \ldots, A_n}$	$(\forall x_1, \ldots, x_p)\ A_1 \wedge \cdots \wedge A_n \Rightarrow A$ where $x_1, \ldots, x_p$ are the variables occurring in $A_1, \ldots, A_n$ and A

where (see Table 1), (Rf) expresses *reflexivity* of many-step rewriting; (Co) expresses *compatibility* of one-step and many-step rewriting; for each k-ary function symbol f and $i \in \mu(f)$, $(\mathrm{Pr})_{f,i}$ enables the *propagation* of rewriting steps in the i-th immediate *active* subterm $t|_i$ of a term t with root symbol f; finally, for each Horn clause $\alpha \in H \cup R$, $(\mathrm{HC})_\alpha$ provides the usual *implicative* form for them. For all terms s and t, we write $s \to_{\mathcal{R}} t$ (resp. $s \to^*_{\mathcal{R}} t$) iff $\overline{\mathcal{R}} \vdash s \to t$ (resp. $\overline{\mathcal{R}} \vdash s \to^* t$). Since $\overline{\mathcal{R}}$ is a Horn theory, the use of, e.g., *resolution* [31] provides a correct and complete proof method for the considered goals.

▶ Remark 2 (Infeasible rules). Only rules $\ell \to r \Leftarrow c \in R$ whose conditional part c is $\overline{\mathcal{R}}$-feasible can be used in rewriting steps. In the following, we assume that all rules in $\mathcal{R}$ can be proved $\overline{\mathcal{R}}$-feasible. Again, resolution can be used for this purpose, see also [13].

The following result provides a *position-based* view of rewriting with GTRSs.

▶ **Proposition 3** ([21, Proposition 58]). *Let $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H, R)$ be a GTRS and $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$. Then, $s \to_{\mathcal{R}} t$ iff there is $p \in \mathcal{P}os^\mu(s)$ and $\ell \to r \Leftarrow c \in R$ such that (i) $s|_p = \sigma(\ell)$ for some substitution σ, (ii) for all $A \in c$, $\overline{\mathcal{R}} \vdash \sigma(A)$ holds, and (iii) $t = s[\sigma(r)]_p$.*

▶ **Definition 4.** *A GTRS $\mathcal{R}$ is terminating iff there is no infinite sequence $t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} \cdots$.*

▶ **Example 5.** Consider the GTRS $\mathcal{R} = (\mathcal{F}, \Pi, \mu_\top, H, R)$ (from COPS #342, [34, Ex. 4])

$$x \approx y \Leftarrow x \to^* y \tag{11}$$
$$\mathsf{f}(x', x'') \to \mathsf{h}(x, \mathsf{f}(x, \mathsf{b})) \Leftarrow x' \approx x, x'' \approx x \tag{12}$$
$$\mathsf{f}(\mathsf{g}(y'), y'') \to \mathsf{h}(y, \mathsf{f}(\mathsf{g}(y), \mathsf{a})) \Leftarrow y' \approx y, y'' \approx y \tag{13}$$
$$\mathsf{a} \to \mathsf{b} \tag{14}$$

where $\mathcal{F} = \{\mathsf{a}, \mathsf{b}, \mathsf{f}, \mathsf{g}, \mathsf{h}\}$, $\Pi = \{\approx, \to, \to^*\}$, $H = \{(11)\}$, and $R = \{(12), (13), (14)\}$. $\overline{\mathcal{R}}$ is:

$$\{(\mathrm{Rf}), (\mathrm{Co}), (\mathrm{Pr})_{\mathsf{f},1}, (\mathrm{Pr})_{\mathsf{f},2}, (\mathrm{Pr})_{\mathsf{g},1}, (\mathrm{Pr})_{\mathsf{h},1}, (\mathrm{Pr})_{\mathsf{h},2}, (\mathrm{HC})_{(11)}, (\mathrm{HC})_{(12)}, (\mathrm{HC})_{(13)}, (\mathrm{HC})_{(14)}\}$$

Note that $\mathcal{R}$ is *not* terminating: $\underline{\mathsf{f}(\mathsf{b}, \mathsf{b})} \to_{\mathcal{R}} \mathsf{h}(\mathsf{b}, \underline{\mathsf{f}(\mathsf{b}, \mathsf{b})}) \to_{\mathcal{R}} \cdots$ We prove it in Example 30.

The following example illustrates the use of replacement maps.

▶ **Example 6.** Consider the GTRS $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H, R)$ with $\mathcal{F}$, Π, H and R as in Example 5 but with $\mu(\mathsf{f}) = \{1, 2\}$, $\mu(\mathsf{g}) = \{1\}$, and $\mu(\mathsf{h}) = \emptyset$. Now,

$$\overline{\mathcal{R}} = \{(\mathrm{Rf}), (\mathrm{Co}), (\mathrm{Pr})_{\mathsf{f},1}, (\mathrm{Pr})_{\mathsf{f},2}, (\mathrm{Pr})_{\mathsf{g},1}, (\mathrm{HC})_{(11)}, (\mathrm{HC})_{(12)}, (\mathrm{HC})_{(13)}, (\mathrm{HC})_{(14)}\}$$

Note the absence of $(\mathrm{Pr})_{\mathsf{h},1}$ and $(\mathrm{Pr})_{\mathsf{h},2}$ due to $\mu(\mathsf{h}) = \emptyset$. Since rewritings on the arguments of h are *forbidden* by $\mu(\mathsf{h}) = \emptyset$, $\mathcal{R}$ is *terminating* (see Example 8 below).

Termination of GTRSs is characterized by interpretation of the symbols as follows.

▶ **Proposition 7** (Termination of GTRSs by interpretation). *A GTRS $\mathcal{R}$ is terminating iff there is a model $\mathcal{A}$ of $\overline{\mathcal{R}}$ with non-empty domain A and $\to^{\mathcal{A}}$ is* terminating *on A.*

▶ **Example 8.** Consider the GTRS $\mathcal{R}$ in Example 6. By Proposition 7, the following model $\mathcal{A}$ of $\overline{\mathcal{R}}$ (computed by the tool AGES [12] for the automatic generation of models of first-order theories) with domain $A = \mathbb{N} \cup \{-1\}$, function symbols interpreted by

$$\mathsf{a}^{\mathcal{A}}=3 \qquad \mathsf{b}^{\mathcal{A}}=0 \qquad \mathsf{f}^{\mathcal{A}}(x)=x+y+13 \qquad \mathsf{g}^{\mathcal{A}}(x)=x \qquad \mathsf{h}^{\mathcal{A}}(x,y)=1$$

and predicate symbols as follows:

$$x \approx^{\mathcal{A}} y \Leftrightarrow \mathit{true} \qquad x \to^{\mathcal{A}} y \Leftrightarrow x > y \qquad x(\to^*)^{\mathcal{A}} y \Leftrightarrow \mathit{true}$$

proves termination of $\mathcal{R}$, as $\to^{\mathcal{A}}$ is terminating on $\mathbb{N} \cup \{-1\}$.

Sometimes, termination of GTRSs $\mathcal{R}$ is difficult to prove by using Proposition 7 due to the need of (automatically) synthesizing appropriate interpretations. In the following, we develop a more powerful approach to prove termination of GTRSs. For instance, in Example 20 we prove termination of $\mathcal{R}$ in Example 6 without synthesizing any interpretation.

4 Infinite rewrite sequences starting from minimal terms

In this section, following [14], we consider the structure of infinite rewrite sequences as the starting point for the subsequent analysis of *termination* of GTRSs.

▶ **Definition 9.** *Let $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H, R)$ be a GTRS. A term t is* nonterminating *if there is an infinite sequence $t = t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} \cdots$. A nonterminating term t is* minimal *if all strict* active *subterms t' of t (i.e., $t \rhd_\mu t'$) are terminating. Let $\mathcal{M}_{\infty,\mu}$ be the set of such minimal nonterminating terms.*

Nonterminating terms contain active minimal nonterminating subterms (cf. [1, Lemma 3]).

▶ **Definition 10.** *Let $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H, R)$ be a GTRS. The set of* active defined subterms *of a term t is $\mathcal{D}_{\trianglerighteq_\mu}(\mathcal{R}, t) = \{t|_p \mid p \in \mathcal{P}os^\mu(t), root(t|_p) \in \mathcal{D}\}$.*

Infinite rewrite sequences starting from a minimal nonterminating term s (i) first rewrite s *below the root* to a term t (written $s \xrightarrow{>\Lambda}{}^*_{\mathcal{R}} t$) which (ii) matches the left-hand side ℓ of a rule $\ell \to r \Leftarrow c \in \mathcal{R}$, i.e., $t = \sigma(\ell)$ for some substitution σ such that $\sigma(c)$ holds, and hence (iii) t rewrites at the root to $\sigma(r)$ (written $t = \sigma(\ell) \xrightarrow{\Lambda}_{\mathcal{R}} \sigma(r)$), and then (iv) $\sigma(r)$ contains a minimal nonterminating subterm u on which the infinite sequence may continue.

▶ **Proposition 11.** *Let $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H, R)$ be a GTRS and $t \in \mathcal{M}_{\infty,\mu}$. There exist $\ell \to r \Leftarrow c \in R$, a substitution σ, and $u \in \mathcal{M}_{\infty,\mu}$ such that $t \xrightarrow{>\Lambda}{}^*_{\mathcal{R}} \sigma(\ell) \xrightarrow{\Lambda}_{\mathcal{R}} \sigma(r) \trianglerighteq_\mu u$, and either*

1. *there is $s \in \mathcal{D}_{\trianglerighteq_\mu}(\mathcal{R}, r)$, $\ell \not\trianglerighteq_\mu s$, such that $u = \sigma(s)$, or*
2. *there is $x \in \mathcal{V}ar^\mu(r) - \mathcal{V}ar^\mu(\ell)$ such that $\sigma(x) \trianglerighteq_\mu u$.*

The following definition collects the rules that are used in the first and second cases of Proposition 11, respectively.

▶ **Definition 12.** *Let $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H, R)$ be a GTRS. Let $DRules(\mathcal{R})$ be the set of rules in $\mathcal{R}$ which* depend *on other defined symbols in $\mathcal{R}$ and $ERules(\mathcal{R})$ be the set of rules in $\mathcal{R}$ whose right-hand sides contain* active *variables which are* not *active in the corresponding left-hand side:*

$$\begin{aligned} DRules(\mathcal{R}) &= \{\ell \to r \Leftarrow c \in \mathcal{R} \mid \mathcal{D}_{\trianglerighteq_\mu}(\mathcal{R}, r) \neq \emptyset\} \\ ERules(\mathcal{R}) &= \{\ell \to r \Leftarrow c \in \mathcal{R} \mid \mathcal{V}ar^\mu(r) - \mathcal{V}ar^\mu(\ell) \neq \emptyset\} \end{aligned}$$

5 Dependency pairs for termination of GTRSs

In the *dependency pair approach* (or just *DP approach*) for TRSs [2], termination of a TRS $\mathcal{R}$ is characterized as the absence of infinite *chains* $\langle u_1, v_1\rangle, \langle u_2, v_2\rangle, \ldots$,[1] where for all $i \geq 1$, $\langle u_i, v_i\rangle$ are *dependency pairs*, which are obtained from the rules $\ell \to r \in \mathcal{R}$ [2, Definition 3] by *marking* the root symbols f (as, e.g., $f^\sharp$ but often *capitalizing* it as F) in the left-hand side ℓ of the rule and also the root symbol of the *defined* subterm s of the right-hand side r which is referred in Proposition 11. Hence, $\langle u, v\rangle$, where $u = \ell^\sharp$ and $v = s^\sharp$ is a *dependency pair* of $\mathcal{R}$. Furthermore, there is a substitution σ such that, for all $i \geq 1$, consecutive pairs are *connected* as follows $\sigma(v_i) \to_{\mathcal{R}}^* \sigma(u_{i+1})$ [2, Definition 5]. This corresponds to the sequence $t \xrightarrow{>\Lambda}_{\mathcal{R}}^* \sigma(\ell)$ referred in Proposition 11. In our generalization of the DP approach to GTRSs $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H, R)$, we use a fresh predicate symbol $\overset{\mathsf{t}}{\to} \notin \Pi$ to represent *conditional pairs* $u \overset{\mathsf{t}}{\to} v \Leftarrow c$ which are then added as new *definite Horn clauses* to H to capture the *termination* behavior of $\mathcal{R}$ (hence the "t" over the arrow). In this way, we obtain a new GTRS $\mathcal{R}'$ whose set of rules is R as well. Since the reduction relation of $\mathcal{R}$ does *not* depend on $\overset{\mathsf{t}}{\to}$, both $\to_{\mathcal{R}}$ and $\to_{\mathcal{R}'}$ *coincide*; thus, we take the pairs from the set of Horn clauses in $\mathcal{R}'$ while using $\to_{\mathcal{R}'}^* = \to_{\mathcal{R}}^*$ to *connect* them.

▶ **Notation 13.** *Given a GTRS $\mathcal{R}$, we write $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H \uplus P, R)$ making explicit the (possibly empty) set P of all clauses $u \overset{\mathsf{t}}{\to} v \Leftarrow c$ for some terms u and v and conditional part c which are used in $\mathcal{R}$. We also write $P_{\mathcal{R}}$ to refer to such a subset P of Horn clauses of $\mathcal{R}$.*

As usual in the DP approach, unless established otherwise, in the following we assume that considered pairs are pairwise variable-disjoint renamings of pairs taken from $P_{\mathcal{R}}$.

▶ **Definition 14** (Chain of pairs of a GTRS). *Let $\mathcal{R}$ be a GTRS. An $\mathcal{R}$-chain is a finite or infinite sequence of pairs $u_i \overset{\mathsf{t}}{\to} v_i \Leftarrow c_i \in P_{\mathcal{R}}$ together with a substitution σ such that, for all $i \geq 1$ ($1 \leq i < n$ for sequences of $n > 1$ pairs), $\overline{\mathcal{R}} \vdash \sigma(c_i), \sigma(v_i) \to^* \sigma(u_{i+1}), \sigma(c_{i+1})$ holds. An $\mathcal{R}$-chain is called* minimal *if for all $i \geq 1$, $\sigma(v_i)$ is terminating.*

According to Definition 14, the existence of a *finite* chain $\alpha_1, \ldots, \alpha_n$ for pairs $\alpha_i : u_i \overset{\mathsf{t}}{\to} v_i \Leftarrow c_i$, $1 \leq i \leq n$, is equivalent to the *feasibility* of the sequence

$$c_1, v_1 \to^* u_2, \ldots, c_{n-1}, v_{n-1} \to^* u_n, c_n \tag{15}$$

▶ Remark 15 (Infeasible pairs). Since only $\overline{\mathcal{R}}$-feasible pairs can be used in $\mathcal{R}$-chains, we can safely *remove* $\overline{\mathcal{R}}$-infeasible pairs from $P_{\mathcal{R}}$ in termination proofs.

As discussed above, we use *marked* symbols $f^\sharp$ associated to (*defined*) symbols f. In general, given a signature $\mathcal{F}$, we let $\mathcal{F}^\sharp = \{f^\sharp \mid f \in \mathcal{F}\}$; and given $\mu \in M_{\mathcal{F}}$ and $\mathcal{D} \subseteq \mathcal{F}$, we let $\mu^\sharp_{\mathcal{D}} \in M_{\mathcal{F} \cup \mathcal{D}^\sharp}$ (or just $\mu^\sharp$ if no confusion arises) be as follows: for all $f \in \mathcal{F} \cup \mathcal{D}^\sharp$,

$$\mu^\sharp(f) = \begin{cases} \mu(f) & \text{if } f \in \mathcal{F} \\ \mu(g) & \text{if } f = g^\sharp \text{ for some } g \in \mathcal{D} \end{cases}$$

As for the DP approach for TRSs, "classical" dependency pairs $u \overset{\mathsf{t}}{\to} v \Leftarrow c$, obtained from rules $\ell \to r \Leftarrow c$ by letting $u = \ell^\sharp$ and $v = s^\sharp$ for some active defined subterm s of v, capture the situation described in the first item of Proposition 11 (see, e.g., [14, Lemma 1] for TRSs).

[1] We use this early notation instead of the current rule-based one, $u \to v$, to prepare the inclusion of our dependency pairs in the component H, rather than R, of the considered GTRS. See Definition 14.

However, dealing with GTRSs, we also need to consider item 2 in Proposition 11 for which "classical" dependency pairs are not appropriate. In this case, Proposition 11 guarantees that a minimal nonterminating subterm s can be found in $\sigma(x)$. In order to *find* and then *mark* s, we use (i) the *active subterm* relation $\trianglerighteq_\mu$ defined by a set of clauses $\mathcal{S}ubt(\mathcal{F},\mu)$ and (ii) a binary predicate symbol Mk defined by a set of clauses $\mathcal{M}ark(\mathcal{F})$ for a signature $\mathcal{F}$ and $\mu \in M_{\mathcal{F}}$:

1. Let $\varpi_{\trianglerighteq_\mu}$ be a new predicate symbol. Then, $\mathcal{S}ubt(\mathcal{F},\mu)$ consists of clauses
$$x\ \varpi_{\trianglerighteq_\mu}\ x \tag{16}$$
$$f(x_1,\ldots,x_i,\ldots,x_k)\ \varpi_{\trianglerighteq_\mu}\ x_i' \quad\Leftarrow\quad x_i\ \varpi_{\trianglerighteq_\mu}\ x_i' \tag{17}$$
for each $f \in \mathcal{F}$, $k = ar(f)$, $i \in \mu(f)$, and variables x and x_i, x_i' for $1 \le i \le k$.
2. Let Mk be a new predicate symbol. Then, $\mathcal{M}ark(\mathcal{F})$ consists of the clauses :
$$\mathsf{Mk}(f(x_1,\ldots,x_k), f^\sharp(x_1,\ldots,x_k)) \tag{18}$$
for each $f \in \mathcal{F}$, $k = ar(f)$, and fresh variables $x_1,\ldots,x_k$.

▶ **Definition 16.** *Let $\mathcal{R} = (\mathcal{F},\Pi,\mu,H,R)$ be a GTRS whose set of defined symbols is $\mathcal{D}$. The GTRS $\mathsf{DP}_{HC}(\mathcal{R})$ is:*

$$(\mathcal{F}\cup\mathcal{D}^\sharp, \Pi\cup\{\varpi_{\trianglerighteq_\mu}, \mathsf{Mk}\}, \mu^\sharp, H\cup\mathcal{S}ubt(\mathcal{F}\cup\mathcal{D}^\sharp,\mu^\sharp)\cup\mathcal{M}ark(\mathcal{D})\cup H_{DP}(\mathcal{R})\cup H_{DPC}(\mathcal{R}), R)$$

where

$$\begin{array}{rcl} H_{DP}(\mathcal{R}) & = & \{\ell^\sharp \xrightarrow{t} v^\sharp \Leftarrow c \mid \ell\to r\Leftarrow c\in DRules(R), v\in\mathcal{D}_{\trianglerighteq_\mu}(\mathcal{R},r), \ell\not\trianglerighteq_\mu v\} \\ H_{DPC}(\mathcal{R}) & = & \{\ell^\sharp \xrightarrow{t} x''\Leftarrow c, x\ \varpi_{\trianglerighteq_\mu}\ x', \mathsf{Mk}(x',x'') \mid \ell\to r\Leftarrow c\in R, x\in\mathcal{V}ar^\mu(r)-\mathcal{V}ar^\mu(\ell) \\ & & \quad \textit{and } x' \textit{ and } x'' \textit{ are fresh variables}\} \end{array}$$

The set of clauses $H_{DP}(\mathcal{R})$ correspond to "classical" conditional dependency pairs for $\mathcal{R}$. $H_{DPC}(\mathcal{R})$ contains *collapsing* dependency pairs for rules $\ell\to r\Leftarrow c$ in $\mathcal{R}$, as the right-hand side of one of such pairs is a *variable* x'' corresponding to a variable $x\in\mathcal{V}ar(r)$ which is active in r but missing, or not active, in ℓ; the conditional part of such pairs uses $\varpi_{\trianglerighteq_\mu}$ to extract active minimal nonterminating subterms s to which x' gets bounded, and finally mark it using Mk so that x'' is finally bound to $s^\sharp$ (see item 2 of Proposition 11). Note that $P_{\mathsf{DP}_{HC}(\mathcal{R})} = H_{DP}(\mathcal{R})\cup H_{DPC}(\mathcal{R})$.

▶ **Example 17.** For $\mathcal{R}$ in Example 1, $P_{\mathsf{DP}_{HC}(\mathcal{R})} = H_{DP}(\mathcal{R})\cup H_{DPC}(\mathcal{R})$ where $H_{DP}(\mathcal{R}) = \{(19)\}$ and $H_{DPC}(\mathcal{R}) = \{(20),(21)\}$, with

$$\mathsf{s}(x) -^\sharp \mathsf{s}(y) \xrightarrow{t} x -^\sharp y \tag{19}$$
$$\mathsf{DIV}(x,y) \xrightarrow{t} q''\Leftarrow y\le x, \mathsf{div}(x-y,y)\approx\mathsf{pair}(q,r), q\ \varpi_{\trianglerighteq_\mu}\ q', \mathsf{Mk}(q',q'') \tag{20}$$
$$\mathsf{DIV}(x,y) \xrightarrow{t} r''\Leftarrow y\le x, \mathsf{div}(x-y,y)\approx\mathsf{pair}(q,r), r\ \varpi_{\trianglerighteq_\mu}\ r', \mathsf{Mk}(r',r'') \tag{21}$$

where, as usual in the DP approach, the uppercase identifier DIV corresponds to $\mathsf{div}^\sharp$. Besides, $\mathcal{S}ubt(\mathcal{F},\mu) = \{(22),\ldots,(33)\}$ and $\mathcal{M}ark(\mathcal{D}) = \{(34),(35)\}$, with

$$x\ \varpi_{\trianglerighteq_\mu}\ x \tag{22}$$
$$s(x)\ \varpi_{\trianglerighteq_\mu}\ x' \Leftarrow x\ \varpi_{\trianglerighteq_\mu}\ x' \tag{23}$$
$$x-y\ \varpi_{\trianglerighteq_\mu}\ x' \Leftarrow x\ \varpi_{\trianglerighteq_\mu}\ x' \tag{24}$$
$$x-y\ \varpi_{\trianglerighteq_\mu}\ y' \Leftarrow y\ \varpi_{\trianglerighteq_\mu}\ y' \tag{25}$$
$$\mathsf{div}(x,y)\ \varpi_{\trianglerighteq_\mu}\ x' \Leftarrow x\ \varpi_{\trianglerighteq_\mu}\ x' \tag{26}$$
$$\mathsf{div}(x,y)\ \varpi_{\trianglerighteq_\mu}\ y' \Leftarrow y\ \varpi_{\trianglerighteq_\mu}\ y' \tag{27}$$
$$\mathsf{pair}(x,y)\ \varpi_{\trianglerighteq_\mu}\ x' \Leftarrow x\ \varpi_{\trianglerighteq_\mu}\ x' \tag{28}$$
$$\mathsf{pair}(x,y)\ \varpi_{\trianglerighteq_\mu}\ y' \Leftarrow y\ \varpi_{\trianglerighteq_\mu}\ y' \tag{29}$$
$$x -^\sharp y\ \varpi_{\trianglerighteq_\mu}\ x' \Leftarrow x\ \varpi_{\trianglerighteq_\mu}\ x' \tag{30}$$
$$x -^\sharp y\ \varpi_{\trianglerighteq_\mu}\ y' \Leftarrow y\ \varpi_{\trianglerighteq_\mu}\ y' \tag{31}$$
$$\mathsf{DIV}(x,y)\ \varpi_{\trianglerighteq_\mu}\ x' \Leftarrow x\ \varpi_{\trianglerighteq_\mu}\ x' \tag{32}$$
$$\mathsf{DIV}(x,y)\ \varpi_{\trianglerighteq_\mu}\ y' \Leftarrow y\ \varpi_{\trianglerighteq_\mu}\ y' \tag{33}$$
$$\mathsf{Mk}(x-y, x-^\sharp y) \tag{34}$$
$$\mathsf{Mk}(\mathsf{div}(x,y),\mathsf{DIV}(x,y)) \tag{35}$$

▶ **Example 18.** For $\mathcal{R}$ in Example 5, $P_{\mathsf{DP}_{HC}(\mathcal{R})} = H_{DP}(\mathcal{R}) \cup H_{DPC}(\mathcal{R})$, where $H_{DP}(\mathcal{R}) = \{(36), (37), (38)\}$ and $H_{DPC}(\mathcal{R}) = \{(39), (40)\}$, with

$$\mathsf{F}(x', x'') \xrightarrow{\mathsf{t}} \mathsf{F}(x, \mathsf{b}) \quad \Leftarrow \quad x' \approx x, x'' \approx x \tag{36}$$

$$\mathsf{F}(\mathsf{g}(y'), y'') \xrightarrow{\mathsf{t}} \mathsf{F}(\mathsf{g}(y), \mathsf{a}) \quad \Leftarrow \quad y' \approx y, y'' \approx y \tag{37}$$

$$\mathsf{F}(\mathsf{g}(y'), y'') \xrightarrow{\mathsf{t}} \mathsf{A} \quad \Leftarrow \quad y' \approx y, y'' \approx y \tag{38}$$

$$\mathsf{F}(x', x'') \xrightarrow{\mathsf{t}} z' \quad \Leftarrow \quad x' \approx x, x'' \approx x, x \mathrel{\varpi_{\trianglerighteq_\mu}} z, \mathsf{Mk}(z, z') \tag{39}$$

$$\mathsf{F}(\mathsf{g}(y'), y'') \xrightarrow{\mathsf{t}} z' \quad \Leftarrow \quad y' \approx y, y'' \approx y, y \mathrel{\varpi_{\trianglerighteq_\mu}} z, \mathsf{Mk}(z, z') \tag{40}$$

▶ **Theorem 19** (Termination). *Let $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H, R)$ be a GTRS.*
1. *$\mathcal{R}$ is* terminating *if there is no infinite minimal $\mathsf{DP}_{HC}(\mathcal{R})$-chain.*
2. *$\mathcal{R}$ is* nonterminating *if there is an infinite $\mathsf{DP}_{HC}(\mathcal{R})$-chain.*

▶ **Example 20** (Termination of $\mathcal{R}$ in Example 6 by absence of $\mathsf{DP}_{HC}(\mathcal{R})$-chains). For $\mathcal{R}$ in Example 6, $H_{DP}(\mathcal{R})$ is empty as the (definite) subterms $\mathsf{f}(x, \mathsf{b})$, $\mathsf{f}(\mathsf{g}(y), \mathsf{a})$, and a in the right-hand sides of rules (13) and (14) are *frozen*. Also, $H_{DPC}(\mathcal{R}) = \emptyset$ as variables x and y in the right-hand sides of rules (13) and (14) are *frozen* too. Thus, no $\mathsf{DP}_{HC}(\mathcal{R})$-chain is possible. By Theorem 19, $\mathcal{R}$ is terminating. In contrast to the proof of termination in Example 8, based on Proposition 7, no synthesis of any model is required here.

The following section introduces a *Dependency Pair Framework* for the *mechanization* of proofs of termination of GTRSs using dependency pairs.

6 The Dependency Pair Framework for GTRSs

The Dependency Pair (DP) Framework is a *divide-and-conquer* technique to prove "termination problems". They are decomposed into *smaller* or *simpler* ones to finally obtain "trivial" problems which can be easily solved. Then, we combine the obtained answers to provide a solution to the initial problem. Thus, the first ingredient of the DP Framework for GTRSs is a suitable notion of *termination problem*. Let $\mathsf{F} = \{\mathsf{a}, \mathsf{m}\}$ be a signature of *flag constants* φ referring to arbitrary (resp. minimal) $\mathcal{R}$-chains if $\varphi = \mathsf{a}$ (resp. $\varphi = \mathsf{m}$).

▶ **Definition 21** (GTRS problem). *A GTRS problem $\tau = (\mathcal{R}, \varphi)$ consists of a GTRS $\mathcal{R}$ and $\varphi \in \mathsf{F}$.*

We often speak of $(\mathcal{R}, \mathsf{m})$-chains (or just τ-chains if $\tau = (\mathcal{R}, \mathsf{m})$) instead of *minimal* $\mathcal{R}$-chains; and of $(\mathcal{R}, \mathsf{a})$-chains (resp. τ-chains if $\tau = (\mathcal{R}, \mathsf{a})$) instead of *arbitrary* $\mathcal{R}$-chains.

▶ **Definition 22** (Finite GTRS problem). *A GTRS problem $\tau = (\mathcal{R}, \varphi)$ is* finite *iff there is no infinite τ-chain; otherwise, τ is* infinite.

Accordingly, the following result rephrases Theorem 19.

▶ **Theorem 23.** *A GTRS $\mathcal{R}$ is terminating iff $(\mathsf{DP}_{HC}(\mathcal{R}), \varphi)$ is finite for some $f \in \mathsf{F}$.*

Processors *transform* GTRS problems $\tau = (\mathcal{R}, \varphi)$ into possibly empty sets $\{\tau_1, \ldots, \tau_n\}$ of GTRS problems $\tau_i = (\mathcal{R}_i, \varphi_i)$, hopefully easier to deal with. Processors may also return "no", with the intended meaning of τ being *infinite*.

▶ **Definition 24** (GTRS processor). *A* GTRS processor P *is a partial function from GTRS problems into sets of GTRS problems. Alternatively, it can return* "no". $\mathsf{Dom}(\mathsf{P})$ *is the set of GTRS problems τ for which P is defined.*

In the following we often speak of "processors" rather than "GTRS processors". For the sake of readability, we often write $\mathsf{P}(\mathcal{R},\varphi)$ rather than $\mathsf{P}((\mathcal{R},\varphi))$. The most relevant properties to be established when using processors are *soundness* and *completeness*. Soundness is essential to prove GTRS problems *finite* by using combinations of processors; completeness for proving *infiniteness*.

▶ **Definition 25** (Soundness and completeness). *Let* P *be a processor and* $\tau \in \mathsf{Dom}(\mathsf{P})$*. We say that* P *is:*

- τ-sound *iff* τ *is finite whenever* $\mathsf{P}(\tau) \neq \mathsf{no}$ *and for all* $\tau' \in \mathsf{P}(\tau)$*,* τ' *is finite.*
- τ-complete *iff* τ *is infinite whenever* $\mathsf{P}(\tau) = \mathsf{no}$ *or there is* $\tau' \in \mathsf{P}(\tau)$ *such that* τ' *is infinite.*

Accordingly, given $\varphi \in \mathsf{F}$*, we say that* P *is* φ-sound *(resp.* φ-complete*) if it is* τ*-sound (*τ*-complete) for all* $\tau = (\mathcal{R},\varphi') \in \mathsf{Dom}(\mathsf{P})$ *such that* $\varphi = \varphi'$*.* P *is* sound *(resp.* complete*) if it is* φ*-sound (*φ*-complete) for all* $\varphi \in \mathsf{F}$*.*

Processors are used in a *divide and conquer* scheme to incrementally simplify a *target* GTRS problem τ_0, possibly decomposing it into (a tree of) smaller problems which are independently treated in the same way.

▶ **Definition 26** (GTRS Proof Tree). *Let* τ_0 *be a GTRS problem. A* GTRS Proof tree $\mathcal{T}$ *(GTRSP-tree for short) for* τ_0 *is a tree whose nodes are labeled with GTRS problems; the leaves may also be labeled with either* "yes" *or* "no"*. The root of* $\mathcal{T}$ *is labeled with* τ_0*. For every inner node* n *with label* τ*, there is a processor* P *such that* $\tau \in \mathsf{Dom}(\mathsf{P})$ *and:*

1. *If* $\mathsf{P}(\tau) = \mathsf{no}$*, then* n *has just one child* n' *with label* "no"*.*
2. *If* $\mathsf{P}(\tau) = \emptyset$*, then* n *has just one child* n' *with label* "yes"*.*
3. *If* $\mathsf{P}(\tau) = \{\tau_1,\ldots,\tau_k\}$ *with* $k > 0$*, then* n *has exactly* k *children* $\mathsf{n}_1,\ldots,\mathsf{n}_k$ *with labels* $\tau_1,\ldots,\tau_k$*, respectively.*

▶ **Theorem 27.** *Let* τ *be a GTRS problem and* $\mathcal{T}$ *be a GTRSP-tree for* τ*. Then,*

1. *If all leaves in* $\mathcal{T}$ *are labeled with "*yes*", and all involved processors are* sound *for the GTRS problems they are applied to, then* τ *is* finite*.*
2. *If* $\mathcal{T}$ *has a leaf with label "*no*" and all processors from* τ *to the leaf are* complete *for the GTRS problems they are applied to, then* τ *is* infinite*.*

Given a GTRS $\mathcal{R}$, $\tau_I = (\mathsf{DP}_{HC}(\mathcal{R}),\varphi)$, where $\varphi \in \mathsf{F}$, is called an *initial problem*, from which a proof of (non)termination of $\mathcal{R}$ is initiated. The specific choice of φ is important in practice, as soundness/completeness of some processor may depend of this choice. However, from an implementation point of view (which we do not address in this paper), we could easily adapt the *open DP framework* discussed in [25, Section 6], which leaves φ unespecified until a particular use of processors establishes some requirements to fix it.

7 Processors for the DP Framework for GTRSs

In this section we introduce several *processors* for their use in proofs of termination of GTRSs and illustrate their application with some examples. Processors P_{Inf} and P_{Cyc} are used to prove GTRS problems infinite (Section 7.1). The *SCC processor* P_{SCC} permits the use of graph techniques to *decompose* GTRS problems (Section 7.2). The *subterm processor* $\mathsf{P}_{\pi,\triangleright_\mu}$ removes pairs from P without paying attention to rules in R (Section 7.3). The *Removal Pair Processor* P_{RP} uses terminating relations to *remove* pairs from P (Section 7.4).

7.1 Proving GTRS problems infinite

The following processor detects a simple kind of infinite GTRS problems.

▶ **Definition 28** (Infiniteness processor). *Let $\mathcal{R}$ be a GTRS and $\varphi \in \mathsf{F}$. Then, P_{Inf} is given by $\mathsf{P}_{Inf}(\mathcal{R}, \varphi) = \mathsf{no}$ iff there is $u \xrightarrow{\mathsf{t}} v \Leftarrow c \in P_{\mathcal{R}}$ and substitutions η and θ such that $\eta(c)$ and $\eta(v) = \theta(\eta(u))$ hold.*

▶ **Theorem 29.** *P_{Inf} is sound and a-complete. If $\tau = (\mathcal{R}, \mathsf{m})$ and v in Definition 28 is ground and contains no symbol from $\mathcal{D}_{\mathcal{R}}$, then P_{Inf} is τ-complete.*

▶ **Example 30** (Nontermination of $\mathcal{R}$ in Example 5). For $\mathsf{DP}_{HC}(\mathcal{R})$ in Example 18, and $\tau_I = (\mathsf{DP}_{HC}(\mathcal{R}), \mathsf{a})$, consider (36) in Example 18, i.e., $\mathsf{F}(x', x'') \xrightarrow{\mathsf{t}} \mathsf{F}(x, \mathsf{b}) \Leftarrow x' \approx x, x'' \approx x$, $\eta = \{x \mapsto \mathsf{b}, x' \mapsto \mathsf{b}, x'' \mapsto \mathsf{b}\}$, and θ be the empty substitution. Since η satisfies the conditional part of (36) and $\eta(\mathsf{F}(x, \mathsf{b})) = \mathsf{F}(\mathsf{b}, \mathsf{b}) = \eta(\mathsf{F}(x', x''))$, we have $\mathsf{P}_{Inf}(\tau_I) = \mathsf{no}$. Hence, by completeness of P_{Cyc} (Theorem 32), τ_I is infinite, and $\mathcal{R}$ is not terminating.

Just requiring (i) *feasibility* of c and that (ii) v matches u in Definition 28 does *not* guarantee completeness. For instance, $\alpha : \mathsf{F}(x) \xrightarrow{\mathsf{t}} \mathsf{F}(\mathsf{s}(x)) \Leftarrow x \to^* \mathsf{0}$ is feasible (use $\sigma(x) = \mathsf{0}$). The right-hand side $\mathsf{F}(\mathsf{s}(x))$ matches the left-hand side $\mathsf{F}(x)$. However, there is no infinite chain using α only. In practice, P_{Inf} will be used with pairs $u \xrightarrow{\mathsf{t}} v$ without conditional part. In this way, checking that v matches u, i.e., $v = \theta(u)$ for some substitution θ, suffices.

▶ **Definition 31** (Cycle processor). *Let $\mathcal{R}$ be a GTRS, and $\varphi \in \mathsf{F}$. Then, P_{Cyc} is given by $\mathsf{P}_{Cyc}(\mathcal{R}, \varphi) = \mathsf{no}$ iff there are $n \geq 1$ pairs $\alpha_1, \ldots, \alpha_n \in P_{\mathcal{R}}$ such that the following sequence is $\overline{\mathcal{R}}$-feasible*

$$c_1, v_1 \to^* u_2, \ldots, c_i, v_i \to^* u_{i+1}, \ldots, c_n, v_n \to^* u_1 \tag{41}$$

▶ **Theorem 32.** *P_{Cyc} is sound and a-complete. If $\tau = (\mathcal{R}, \mathsf{m})$ and for all $1 \leq i \leq n$ and v_i in pairs α_i in Definition 31 is ground and contains no symbol from $\mathcal{D}_{\mathcal{R}}$, then P_{Cyc} is τ-complete.*

▶ **Example 33.** Consider $\mathcal{R} = (\mathcal{F}, \Pi, \mu_\top, H, R)$, where $H = \{(42)\}$ and $R = \{(43), (44)\}$:

$$x \approx y \;\Leftarrow\; x \to^* y \qquad (42) \qquad\qquad \mathsf{b} \;\to\; \mathsf{a} \qquad (44)$$

$$\mathsf{a} \to \mathsf{c}(x) \;\Leftarrow\; x \approx \mathsf{b} \qquad (43)$$

Regarding $\mathsf{DP}_{HC}(\mathcal{R})$, $\mathcal{S}ubt(\mathcal{F}, \mu)$ and $\mathcal{M}ark(\mathcal{F})$ are defined as explained above, and we have $H_{DP}(\mathcal{R}) = \{(45)\}$ and $H_{DPC}(\mathcal{R}) = \{(46)\}$, with

$$\mathsf{B} \;\to\; \mathsf{A} \tag{45}$$

$$\mathsf{A} \;\to\; x'' \Leftarrow x \approx \mathsf{b}, x \varpi_{\trianglerighteq_\mu} x', \mathsf{Mk}(x', x'') \tag{46}$$

We prove $\mathcal{R}$ nonterminating by applying P_{Cyc} to $\tau_I = (\mathsf{DP}_{HC}(\mathcal{R}), \mathsf{a})$. The following sequence built using the pairs in $H_{DP}(\mathcal{R})$ and $H_{DPC}(\mathcal{R})$ according to (41):

$$\mathsf{A} \to^* \mathsf{A}, x \approx \mathsf{b}, x \varpi_{\trianglerighteq_\mu} x', \mathsf{Mk}(x', x''), x'' \to^* \mathsf{B} \tag{47}$$

is $\overline{\mathsf{DP}_{HC}(\mathcal{R})}$-feasible, as it is satisfied by $\sigma = \{x \mapsto \mathsf{b}, x' \mapsto \mathsf{b}, x'' \mapsto \mathsf{B}\}$. Thus, $\mathsf{P}_{Cyc}(\tau_I) = \mathsf{no}$.

Note that P_{Inf} does *not* apply to $\mathcal{R}$ in Example 33: the only substitution satisfying the conditional part of (46) is σ in the example. But the instance $\sigma(x'') = \mathsf{B}$ of the right-hand side of (46) does *not* match the left-hand side A of (46). However, P_{Cyc} does *not* subsume P_{Inf}. For instance, $\mathsf{F}(x) \xrightarrow{\mathsf{t}} \mathsf{F}(\mathsf{s}(x))$, where s is not a defined symbol, can be used to define an infinite chain which would be easily detected by P_{Inf}. However, $\mathsf{F}(\mathsf{s}(x)) \to^* \mathsf{F}(x)$ is infeasible and P_{Cyc} could not be used to detect infiniteness of a GTRS-problem involving such a pair.

7.2 The SCC processor

In this section we provide a notion of graph which captures infinite (*minimal*) chains of (dependency) pairs as given in Definition 14.

▶ **Definition 34** (GTRS Graph of Pairs). *Let $\mathcal{R}$ be a GTRS. The* GTRS-graph $\mathsf{G}(\mathcal{R})$ *has $P_{\mathcal{R}}$ as the set of nodes. There is an arc from $\alpha \in P_{\mathcal{R}}$ to $\alpha' \in P_{\mathcal{R}}$ iff α, α' is an $\mathcal{R}$-chain for some substitution σ.*

The following result *approximates* the (in general incomputable) dependency graph using the *infeasibility* of some sequences. Although (in)feasibility is undecidable, a number of techniques and tools have been developed to automatically (dis)prove it, thus providing a practical approach to *approximate* the graph.

▶ **Definition 35** (Estimated GTRS Graph). *Let $\mathcal{R}$ be a GTRS. The* estimated *GTRS-graph* $\mathsf{EG}(\mathcal{R})$ *has $P_{\mathcal{R}}$ as the set of nodes. Let $\alpha : u \xrightarrow{\mathsf{t}} v \Leftarrow c, \alpha' : u' \xrightarrow{\mathsf{t}} v' \Leftarrow c' \in P_{\mathcal{R}}$ be such that $\mathcal{V}ar(\alpha) \cap \mathcal{V}ar(\alpha') = \emptyset$ (rename if necessary). There is an arc from α to α' in* $\mathsf{EG}(\mathcal{R})$ *iff $c, v \to^* u', c'$* cannot *be proved $\overline{\mathcal{R}}$-infeasible.*

The sequence considered in Definition 35 actually characterizes the GTRS chain required in Definition 34 to draw an arc from a node α_1 to a node α_2 (see (15)): there is such an arc iff $c, c', v \to^* u'$ is feasible. Definition 35 provides an *estimation* as only an *$\overline{\mathcal{R}}$-infeasibility proof* is attempted (by using some method, tool, etc.); if it succeeds, the arc is *dismissed*; otherwise, the arc is *included* in the graph. Thus, all nodes and arcs of $\mathsf{G}(\mathcal{R})$ are in $\mathsf{EG}(\mathcal{R})$.

▶ **Example 36.** Consider the GTRS $\mathcal{R}$ in Example 1 and $\mathsf{DP}_{HC}(\mathcal{R})$ in Example 17. The estimated dependency graph $\mathsf{EG}(\mathsf{DP}_{HC}(\mathcal{R}))$ is

For instance, the following sequence, that corresponds to an arc from (20) to (20),

$$y_1 \le x_1, \mathsf{div}(x_1 - y_1, y_1) \approx \mathsf{pair}(q_1, r_1), q_1 \varpi_{\rhd_\mu} q_1', \mathsf{Mk}(q_1', q_1''),$$
$$y_2 \le x_2, \mathsf{div}(x_2 - y_2, y_2) \approx \mathsf{pair}(q_2, r_2), q_2 \varpi_{\rhd_\mu} q_2', \mathsf{Mk}(q_2', q_2''), q_1'' \to_{\mathcal{R}}^* \mathsf{DIV}(x_2, y_2)$$

can be proved $\overline{\mathsf{DP}_{HC}(\mathcal{R})}$-infeasible (see [13] for a general treatment of such infeasibility problems, including mechanization issues).

▶ **Definition 37** (SCC processor). *Let $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H \uplus P, R)$ be a GTRS and $\tau = (\mathcal{R}, \varphi)$ be an $\mathcal{R}$-problem. Then, P_{SCC} is given by*

$$\mathsf{P}_{SCC}(\tau) = \{((\mathcal{F}, \Pi, \mu, H \uplus P', R), \varphi) \mid P' \textit{are the nodes of an SCC in } \mathsf{EG}(\mathcal{R})\}$$

▶ **Example 38.** For $\mathcal{R}_0 = \mathsf{DP}_{HC}(\mathcal{R}) = (\mathcal{F}, \Pi, \mu^\sharp, H \uplus \{(19), (20), (21)\}, R)$ in Example 17 and $\mathsf{EG}(\mathcal{R}_0)$ displayed in Example 36, we have $\mathsf{P}_{SCC}(\mathcal{R}_0, \varphi) = \{\tau_1\}$, where $\tau_1 = (\mathcal{R}_1, \varphi)$, with $\mathcal{R}_1 = (\mathcal{F}, \Pi, \mu^\sharp, H \uplus \{(19)\}, R)$.

▶ **Theorem 39.** *P_{SCC} is sound and complete.*

With P_{SCC} we can *separately* work with the strongly connected components of $\mathsf{EG}(\mathcal{R}, \varphi)$, disregarding other parts of the graph.

7.3 Subterm processor

This section generalizes the *subterm processor* for TRSs [14] to GTRSs $\mathcal{R}$. Such a processor removes pairs $u \xrightarrow{\mathsf{t}} v \Leftarrow c \in P_{\mathcal{R}}$ if some immediate subterm $v|_j$ of v is a *strict* active subterm of an immediate subterm $u|_i$ of u, i.e., $u|_i \rhd_\mu v|_j$ holds, where i and j are determined by the root symbols of u and v, respectively, by means of a so-called *simple projection*. The processor does *not* pay attention to the conditional part c of α or to the rules in R. The set of *root symbols* associated to $P_{\mathcal{R}}$ is:

$$Root(P_{\mathcal{R}}) \quad = \quad \{root(u) \mid u \xrightarrow{\mathsf{t}} v \Leftarrow c \in P_{\mathcal{R}}\} \cup \{root(v) \mid u \xrightarrow{\mathsf{t}} v \Leftarrow c \in P_{\mathcal{R}}, v \notin \mathcal{X}\}$$

▶ **Definition 40** (Simple projection). *Let $\mathcal{R}$ be a GTRS. A* simple projection *for $\mathcal{R}$ is a mapping $\pi : Root(P_{\mathcal{R}}) \to \mathbb{N}$ such that $\pi(f) \in \{1, \ldots, ar(f)\}$. The mapping that assigns a subterm $\pi(t) = t|_{\pi(f)}$ to each term t with $root(t) \in Root(P_{\mathcal{R}})$ is also denoted by π; we also let $\pi(x) = x$ if $x \in \mathcal{X}$.*

▶ **Definition 41** (Subterm processor). *Let $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H \uplus P, R)$ be a GTRS, π be a simple projection for $\mathcal{R}$, and $\alpha : u \xrightarrow{\mathsf{t}} v \Leftarrow c \in P$. Then, $\mathsf{P}_{\pi,\rhd_\mu}$ is given by $\mathsf{P}_{\pi,\rhd_\mu}(\mathcal{R}, \varphi) = \{(\mathcal{F}, \Pi, \mu, H \uplus (P - \{\alpha\}), R)\}$, if (i) for all $u' \xrightarrow{\mathsf{t}} v' \Leftarrow c' \in P - \{\alpha\}$ we have that $\pi(u') \trianglerighteq_\mu \pi(v')$ holds and (ii) $\pi(u) \rhd_\mu \pi(v)$ holds.*

Note that $\mathsf{P}_{\pi,\rhd_\mu}$ can *not* be applied to a GTRS problem $(\mathcal{R}, \varphi)$ if $P_{\mathcal{R}}$ contains pairs $u \xrightarrow{\mathsf{t}} v \Leftarrow c$ from $H_{DPC}(\mathcal{R})$. This is because, by definition, v is a fresh variable *not* belonging to u. Thus, neither (i) nor (ii) in Definition 41 are fulfilled and $(\mathcal{R}, \varphi) \notin \mathsf{Dom}(\mathsf{P}_{\pi,\rhd_\mu})$.

▶ **Theorem 42** (Soundness and completeness of $\mathsf{P}_{\pi,\rhd_\mu}$). *$\mathsf{P}_{\pi,\rhd_\mu}$ is complete and τ-sound for all $\tau = (\mathcal{R}, \varphi) \in \mathsf{Dom}(\mathsf{P}_{\pi,\rhd_\mu})$ such that for all $u \xrightarrow{\mathsf{t}} v \Leftarrow c \in P_{\mathcal{R}}$ $v \notin \mathcal{X}$, $\varphi = \mathsf{m}$, and $Root_P(\mathcal{R}) \cap \mathcal{D}_{\mathcal{R}} = \emptyset$.*

▶ **Example 43** (Termination of $\mathcal{R}$ in Example 1). Consider $\tau_1 = (\mathcal{R}_1, \varphi)$ in Example 38 with $P_{\mathcal{R}_1} = \{(19)\}$, where (19) is $\mathsf{s}(x) -^\sharp \mathsf{s}(y) \xrightarrow{\mathsf{t}} x -^\sharp y$. Since $Root(P_{\mathcal{R}_1}) = \{-^\sharp\}$, with $\pi(-^\sharp) = 1$ we obtain $\pi(\mathsf{s}(x) -^\sharp \mathsf{s}(y)) = \mathsf{s}(x) \rhd_\mu x = \pi(x -^\sharp y)$. Hence, $\mathsf{P}_{\pi,\rhd_\mu}(\tau_1) = \{(\mathcal{R}_{11}, \varphi)\}$ where $P_{\mathcal{R}_{11}} = \emptyset$. This proves termination of $\mathcal{R}$ in Example 1, after a "formal" application of P_{SCC} to obtain an empty set of GTRS problems, see Figure 1 left.

7.4 Use of terminating relations

The absence of infinite $\mathcal{R}$-chains can be ensured by using terminating relations on terms.

▶ **Definition 44** (Removal pair). *A* removal pair $(\gtrsim, \sqsupset)$ *consists of relations $\gtrsim$ and $\sqsupset$ on terms such that $\sqsupset$ is terminating and $\sqsupset \circ \gtrsim \subseteq \sqsupset$.*

▶ **Definition 45** (Compatible removal pair). *Let $\mathcal{R}$ be a GTRS. A removal pair $(\gtrsim, \sqsupset)$ is* compatible *with $\mathcal{R}$ if (i) $\to_{\mathcal{R}}^* \subseteq \gtrsim$ and (ii) if $P_{\mathcal{R}}$ is not a singleton, then for all $u \xrightarrow{\mathsf{t}} v \Leftarrow c, u' \xrightarrow{\mathsf{t}} v' \Leftarrow c' \in P_{\mathcal{R}}$ and for all substitutions σ, if $\overline{\mathcal{R}} \vdash \sigma(c), \sigma(v) \to^* \sigma(u'), \sigma(c')$ holds, then (ii.1) $\sigma(u) \gtrsim \sigma(v)$ or (ii.2) $\sigma(u) \sqsupset \sigma(v)$.*

Compatibility guarantees that, if α is followed by another (possibly the same, up to renaming) pair α' in a chain, then, by (i), a non-strict decrease is introduced in the transition $\sigma(v) \to_{\mathcal{R}}^* \sigma(u')$, and, if $P_{\mathcal{R}}$ is not a singleton, then there is a non-strict (ii.1) or a strict (ii.2) decrease in the instances of components u and v of α. In the second case (ii.2), we can *remove* α. This is the purpose of the removal pair processor. If $P_{\mathcal{R}}$ is a singleton, then (ii.1) is useless when the processor is applied to remove the only pair in $P_{\mathcal{R}}$. In this case, (ii.2) is specifically required by the processor, as shown in the following.

▶ **Definition 46** (Removal pair processor). *Let* $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H \uplus P, R)$ *be a GTRS,* $\alpha : u \xrightarrow{\mathsf{t}} v \Leftarrow c \in P$*, and* $(\gtrsim, \sqsupset)$ *be a removal pair compatible with* $\mathcal{R}$*. Then,* $\mathsf{P}_{RP}(\mathcal{R}, \varphi) = \{((\mathcal{F}, \Pi, \mu, H \uplus (P - \{\alpha\}), R), \varphi)\}$ *if for all substitutions* σ*, whenever* $\overline{\mathcal{R}} \vdash \sigma(c)$ *holds, we have* $\sigma(u) \sqsupset \sigma(v)$*.*

▶ **Theorem 47** (Soundness and completeness of P_{RP}). P_{RP} *is sound and complete.*

As discussed in [18, 26], the following *semantic* approach is useful in practice.

▶ **Definition 48** (Semantic version of the removal pair processor). *Let* $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H \uplus P, R)$ *be a GTRS and* $\alpha : u \xrightarrow{\mathsf{t}} v \Leftarrow c \in P$*. Let* $\varpi_{\gtrsim}$ *and* $\varpi_{\sqsupset}$ *be fresh predicate symbols. Let* $\mathcal{A}$ *be a model of* $\overline{\mathcal{R}}$ *such that* $\pi^{\mathcal{A}}_{\sqsupset}$ *is* terminating *on the domain of* $\mathcal{A}$*. Then,* $\mathsf{P}_{RP}(\mathcal{R}, \varphi) = \{((\mathcal{F}, \Pi, \mu, H \uplus (P - \{\alpha\}), R), \varphi)\}$ *if each of the following conditions hold:*

1. $\mathcal{A} \models (\forall x, y, z)\ ((x\ \varpi_{\sqsupset}\ y \wedge y\ \varpi_{\gtrsim}\ z) \Rightarrow x\ \varpi_{\sqsupset}\ z)$*,*
2. $\mathcal{A} \models (\forall x, y)\ (x\ \to^*\ y \Rightarrow x\ \varpi_{\gtrsim}\ y)$*,*
3. *if* P *is not a singleton, then for all* $u' \xrightarrow{\mathsf{t}} v' \Leftarrow c', u'' \xrightarrow{\mathsf{t}} v'' \Leftarrow c'' \in P$*,* $\mathcal{A} \models (\forall \vec{x})((c' \wedge v' \to^* u'' \wedge c'') \Rightarrow (u'\ \varpi_{\gtrsim}\ v' \ \vee\ u'\ \varpi_{\sqsupset}\ v'))$*,*
4. $\mathcal{A} \models (\forall \vec{x})\ (c \Rightarrow u\ \varpi_{\sqsupset}\ v)$*.*

The removal pair $(\gtrsim, \sqsupset)$ implicit in Definition 48 is as follows: for all terms s and t, $s \gtrsim t$ (resp. $s \sqsupset t$) iff for all valuations $\nu : \mathcal{X} \to \mathcal{A}$, $[s]^{\mathcal{A}}_{\nu}\ \varpi^{\mathcal{A}}_{\gtrsim}\ [t]^{\mathcal{A}}_{\nu}$ (resp. $[s]^{\mathcal{A}}_{\nu}\ \varpi^{\mathcal{A}}_{\sqsupset}\ [t]^{\mathcal{A}}_{\nu}$) holds, where for all terms $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $[t]^{\mathcal{A}}_{\nu}$ denotes the (usual) interpretation of terms t using the interpretations $f^{\mathcal{A}}$ of function symbols $f \in \mathcal{F}$ by $\mathcal{A}$ and the interpretation $\nu(x)$ of variables x by ν. Thus, requirement (1) in Definition 48 guarantees that $\sqsupset \circ \gtrsim\ \subseteq\ \sqsupset$ in Definition 44 holds; (2) and (3) make $(\gtrsim, \sqsupset)$ compatible with $\mathcal{R}$ (Definition 45); and (4) permits the removal of α (Definition 46).

▶ **Example 49.** Consider the following GTRS $\mathcal{R}$ (COPS #330 [28, Ex. 5.1]):

$$x \approx y \quad \Leftarrow \quad x \to^* y \tag{48}$$

$$\mathsf{pin}(x) \to \mathsf{pout}(\mathsf{g}(x)) \tag{49}$$

$$\mathsf{pin}(x) \to \mathsf{pout}(\mathsf{f}(y)) \quad \Leftarrow \quad \mathsf{pin}(x) \approx \mathsf{pout}(\mathsf{g}(y)) \tag{50}$$

In $\mathsf{DP}_{HC}(\mathcal{R})$, we have $H_{DP}(\mathcal{R}) = \emptyset$; and $H_{DPC}(\mathcal{R})$ consists of a single clause:

$$\mathsf{PIN}(x) \xrightarrow{\mathsf{t}} y'' \quad \Leftarrow \quad \mathsf{pin}(x) \approx \mathsf{pout}(\mathsf{g}(y)), y\ \varpi_{\trianglerighteq_\mu}\ y', \mathsf{Mk}(y', y'') \tag{51}$$

With P_{RP} we can remove (51). Since $P_{\mathsf{DP}_{HC}(\mathcal{R})} = \{(51)\}$ is a singleton, requirement (3) in Definition 48 is not necessary. Requirements (1), (2), and (4) amount at considering:

$$(\forall x, y, z) \qquad x\ \varpi_{\sqsupset}\ y \wedge y\ \varpi_{\gtrsim}\ z \Rightarrow x\ \varpi_{\sqsupset}\ z \tag{52}$$

$$(\forall x, y) \qquad x \to^* y \Rightarrow x\ \varpi_{\gtrsim}\ y \tag{53}$$

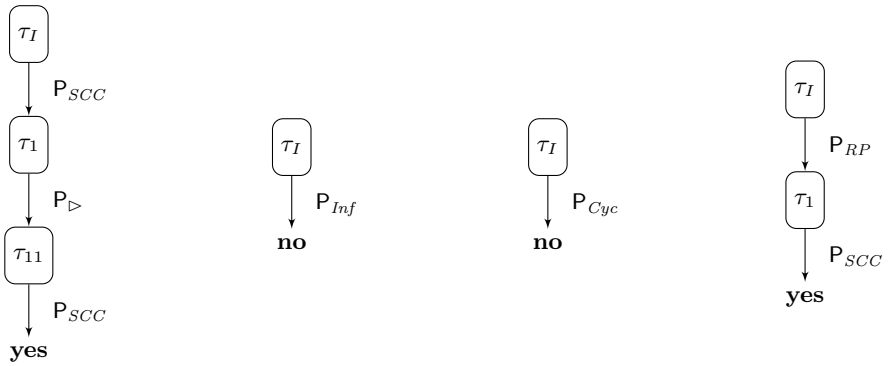
$$(\forall x, y, y', y') \qquad (\mathsf{pin}(x) \approx \mathsf{pout}(\mathsf{g}(y)) \wedge y\ \varpi_{\trianglerighteq_\mu}\ y' \wedge \mathsf{Mk}(y', y'')) \Rightarrow \mathsf{PIN}(x)\ \varpi_{\sqsupset}\ y'' \tag{54}$$

The following structure $\mathcal{A}$ (computed by AGES) with domain $\mathbb{Z} - \mathbb{N}$ (the set of nonpositive integers); function symbols are interpreted as follows:

$$\mathsf{f}^{\mathcal{A}}(x){=}2x \qquad \mathsf{g}^{\mathcal{A}}(x){=}2x \qquad \mathsf{pin}^{\mathcal{A}}(x){=}2x - 1 \qquad \mathsf{pout}^{\mathcal{A}}(x){=}x - 1 \qquad \mathsf{PIN}^{\mathcal{A}}(x){=}2x$$

and predicate symbols as follows:

$$x\ \approx^{\mathcal{A}}\ y \Leftrightarrow y \geq x \qquad x\ \to^{\mathcal{A}}\ y \Leftrightarrow y \geq x \qquad x\ (\to^*)^{\mathcal{A}}\ y \Leftrightarrow y \geq x$$
$$x\ (\xrightarrow{\mathsf{t}})^{\mathcal{A}}\ y \Leftrightarrow \mathit{true} \qquad x\ \varpi^{\mathcal{A}}_{\trianglerighteq_\mu}\ y \Leftrightarrow y \geq x \qquad \mathsf{Mk}^{\mathcal{A}}(x, y) \Leftrightarrow y > x$$
$$x\ \varpi^{\mathcal{A}}_{\gtrsim}\ y \Leftrightarrow y \geq x \qquad x\ \varpi^{\mathcal{A}}_{\sqsupset}\ y \Leftrightarrow y > x$$



Examples 1, 38 & 43 | Examples 5 & 30 | Example 33 | Example 49

Figure 1 Proofs of (non)termination in the DP Framework for the running examples.

is a model of $\overline{\mathsf{DP}_{HC}(\mathcal{R})} \cup \{(52), (53), (54)\}$. Hence (51) can be removed from $\mathsf{DP}_{HC}(\mathcal{R})$ to obtain a GTRS $\mathcal{R}_1$, with $P_{\mathcal{R}_1} = \emptyset$, i.e., $\mathsf{P}_{RP}(\mathsf{DP}_{HC}(\mathcal{R}), \varphi) = \{\tau_1\}$, where $\tau_1 = (\mathcal{R}_1, \varphi)$. Now, $\mathsf{P}_{SCC}(\mathcal{R}_1, \varphi) = \emptyset$ (due to $P_{\mathcal{R}_1} = \emptyset$) completes the proof.

The proof trees for our running examples are shown in Figure 1.

8 Related work

The results in this paper extend and generalize the treatment of termination of *oriented* CTRSs presented in [24] which is implemented using the 2D DP Framework for oriented (2-)CTRSs [25, 26]. Furthermore, some improvements on [25, 26] are obtained:

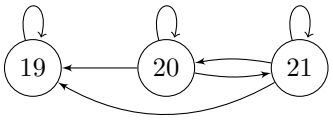
- In contrast to [24], Definition 16 provides a uniform definition of $\mathsf{DP}_{HC}(\mathcal{R})$ with pairs of two classes, $H_{DP}(\mathcal{R})$ and $H_{DPC}(\mathcal{R})$, but with a single definition of $\mathcal{R}$-chain. Also, the use of atoms $x \varpi_{\trianglerighteq_\mu} x'$ and $\mathsf{Mk}(x', x'')$ in the conditional part of pairs in $H_{DPC}(\mathcal{R})$ (also in contrast to [24, Definition 41]) often helps to prove $\overline{\mathsf{DP}_{HC}(\mathcal{R})}$-infeasibility of pairs in $H_{DPC}(\mathcal{R})$, which is helpful. For instance, (20) and (21) are actually $\overline{\mathsf{DP}_{HC}(\mathcal{R})}$-infeasible for $\mathcal{R}$ in Example 1. This would not happen with the corresponding pairs obtained from [24, Definition 41]. We included (20) and (21) here just to illustrate the use of P_{SCC}. A simpler proof would be obtained otherwise.
- Proofs of termination of CTRSs in the 2D DP Framework are restricted, in practice, to 2-CTRSs, i.e., those whose rules $\ell \to r \Leftarrow c$ satisfy that all variables in r also occur in ℓ [25, Theorem 16(1)].[2] Neither the CTRSs $\mathcal{R}$ in Examples 5 and 49 nor Example 1, viewed as a CTRS, are 2-CTRSs.
- As a consequence of [25, Theorem 16(1)], only processors which are sound on *arbitrary* chains can be used in proofs of termination in the 2D DP Framework for CTRSs. For instance, $\mathsf{P}_{\triangleright}$ cannot be used, as it is sound on minimal chains only. In contrast, our termination proof of $\mathcal{R}$ in Example 1 uses $\mathsf{P}_{\triangleright}$.
- P_{Cyc} was not considered in [25, 26] and can be used to detect infinite chains involving an arbitrary number of pairs. Also, P_{Cyc} can be implemented as a *feasibility problem*.

[2] Actually, [25, Theorem 16(1)] uses the more general requirement of rules that *preserve terminating substitutions* discussed in [24, Section 4.2]. A sufficient condition guaranteeing the property is being a 2-CTRS [24, Proposition 37].

```
mod Ex9_SchGra10 is
 sorts Boolean S .
 op true : -> Boolean .                  op 0 : -> S .       op s : S -> S .
 ops _>_ : S S -> Boolean [frozen] .     op _<=_ : S S -> Boolean [frozen] .
 op _-_ : S S -> S .                     op pair : S S -> S .
 op div : S S -> S .                     var q r x y : S .
 eq s(x) > 0 = true .          eq s(x) > s(y) = x > y .   eq 0 <= x = true .
 eq s(x) <= s(y) = x <= y .    rl x - 0 => x .
 rl 0 - y => y .               rl s(x) - s(y) => x - y .
 crl div(x,y) => pair(0,x) if y > x = true .
 crl div(x,y) => pair(s(q),r) if y <= x = true /\ div(x - y,y) => pair(q,r) .
endm
```

Figure 2 Maude enconding of the GTRS $\mathcal{R}$ in Example 1.

Approximations of the dependency graph for CTRSs in [25, Section 7.1.1] pay *no attention* to the conditional parts of pairs [25, Definitions 55 & 56]. For instance, for $\mathsf{DP}_{HC}(\mathcal{R})$ in Example 17, instead of the graph in Example 36, the techniques in [25] produce:



GTRSs also include CS-CTRS [19, Section 8.1], for which no technique for proving termination is known. Also, computations with Generalized Rewrite Theories (GRTs [4]) and Maude programs [5, 6] can often be simulated using (Equational) GTRSs as briefly discussed in [20, 21]. Vice versa, Maude can be used as a practical platform to use GTRSs.

▶ **Example 50.** The Maude module in Figure 2 provides an appropriate translation of $\mathcal{R}$ in Example 1 to Maude. Predicates are treated as *boolean* functions defined by *equations* obtained from the corresponding definite Horn clauses. Also note that the arguments of "predicate" function symbols are all *frozen*, thus disabling reductions on their arguments. This makes Maude computations with `Ex9_SchGra10` closer to $\mathcal{R}$, viewed as a GTRS. For instance, the result of the integer division of 3 by 2 is computed as follows:

```
Maude> rew div(s(s(s(0))),s(s(0))) .
rewrite in Ex9_SchGra10 : div(s(s(s(0))), s(s(0))) .
rewrites: 12 in 0ms cpu (0ms real) (363636 rewrites/second)
result S: pair(s(0), s(0))
```

Thus, our results provide a basis for the development of techniques and tools for proving termination of GRTs and Maude programs.

Finally, at first sight, our techniques would also apply to prove termination of *Logically Constrained TRSs* (LCTRSs [15, 16]). LCTRSs and GTRSs differ in the treatment of the conditional part of the rules: LCTRSs use fixed interpretations and GTRSs additional Horn clauses. Although rewriting with LCTRSs (which requires substitutions that *respect* the rules [15, page 347]) is, in general, more restrictive than rewriting with GTRSs, our notion of dependency pair would capture termination of LCTRSs, although the treatment of the DP problems would be different due to the semantic treatment of conditions.

9 Conclusions and future work

We have presented a Dependency Pair Framework for (dis)proving termination of GTRSs. To the best of our knowledge, this is the first time that the DP Framework has been adapted to deal with GTRSs. It extends previous proposals for proving termination of CTRSs [25]. As discussed in the previous section, even when applied to CTRSs (as particular GTRSs), we are able to obtain proofs of termination which are not possible in the 2D DP Framework. Also, the proof of termination of the CTRS $\mathcal{R}$ in Example 5 supplied with a replacement map μ (see Example 8) cannot be obtained either in the DP Framework for CSR (as it does not consider conditional rules) or in the 2D DP Framework (replacement maps are not allowed).

From a theoretical point of view, a lot of work remains to be done. For instance, the refinements in the treatement of frozen subterms in the right-hand sides of unconditional rules developed in [10, 11] should be adapted to the new conditional setting. Also, the improvements on the reduction pair processor obtained by considering powerful notions such as that of *usable rule*, which are already in use both for CS-TRSs [11, Section 6] and CTRSs [26, Section 4.4] should also be developed for GTRSs and the DP Framework introduced here. Also, extending the framework to deal with *operational termination* of GTRSs is also important, as operational termination provides the best conditions to compute with rewriting-based systems using conditional rules (see [23]). The approach for CTRSs developed in [25] (dealing with the *vertical* dimension of operational termination by means of an additional set of dependency pairs) should be thoroughly revised for GTRSs $\mathcal{R}$ as operational termination of GTRSs depends on proofs of atoms using the Horn theory component H of $\mathcal{R}$.

The *implementation* of the DP Framework for proving termination of GTRSs introduced here is also an important subject for future work. Regarding the implementation of our techniques, we are improving our tool MU-TERM [13] with the ability to deal with GTRSs. However, this actually depends on other tools, in particular infChecker [13], still unable to deal with GTRSs. Also, as discussed in [21], GTRSs can be useful to investigate computational properties of GRTs and Maude programs, which implement GRTs.

References

1 Beatriz Alarcón, Raúl Gutiérrez, and Salvador Lucas. Context-sensitive dependency pairs. *Inf. Comput.*, 208(8):922–968, 2010. doi:10.1016/j.ic.2010.03.003.

2 Thomas Arts and Jürgen Giesl. Termination of term rewriting using dependency pairs. *Theor. Comput. Sci.*, 236(1-2):133–178, 2000. doi:10.1016/S0304-3975(99)00207-8.

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

4 Roberto Bruni and José Meseguer. Semantic foundations for generalized rewrite theories. *Theor. Comput. Sci.*, 360(1-3):386–414, 2006. doi:10.1016/j.tcs.2006.04.012.

5 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 *Lecture Notes in Computer Science*. Springer, 2007. doi:10.1007/978-3-540-71999-1.

6 Francisco Durán, Steven Eker, Santiago Escobar, Narciso Martí-Oliet, José Meseguer, Rubén Rubio, and Carolyn L. Talcott. Programming and symbolic computation in Maude. *J. Log. Algebr. Meth. Program.*, 110, 2020. doi:10.1016/j.jlamp.2019.100497.

7 Melvin Fitting. *First-Order Logic and Automated Theorem Proving, Second Edition.* Graduate Texts in Computer Science. Springer, 1996. doi:10.1007/978-1-4612-2360-3.

8 Jürgen Giesl, René Thiemann, and Peter Schneider-Kamp. The dependency pair framework: Combining techniques for automated termination proofs. In Franz Baader and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 11th International Conference, LPAR 2004, Proceedings*, volume 3452 of *Lecture Notes in Computer Science*, pages 301–331. Springer, 2004. doi:10.1007/978-3-540-32275-7_21.

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

10 Raúl Gutiérrez and Salvador Lucas. Proving termination in the context-sensitive dependency pair framework. In Peter Csaba Ölveczky, editor, *Rewriting Logic and Its Applications - 8th International Workshop, WRLA 2010, Held as a Satellite Event of ETAPS 2010, Revised Selected Papers*, volume 6381 of *Lecture Notes in Computer Science*, pages 18–34. Springer, 2010. doi:10.1007/978-3-642-16310-4_3.

11 Raúl Gutiérrez and Salvador Lucas. Function calls at frozen positions in termination of context-sensitive rewriting. 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 *Lecture Notes in Computer Science*, pages 311–330. Springer, 2015. doi:10.1007/978-3-319-23165-5_15.

12 Raúl Gutiérrez and Salvador Lucas. Automatic Generation of Logical Models with AGES. In Pascal Fontaine, editor, *Automated Deduction - CADE 27 - 27th International Conference on Automated Deduction, Proceedings*, volume 11716 of *Lecture Notes in Computer Science*, pages 287–299. Springer, 2019. doi:10.1007/978-3-030-29436-6_17.

13 Raúl Gutiérrez and Salvador Lucas. Automatically Proving and Disproving Feasibility Conditions. In Nicolas Peltier and Viorica Sofronie-Stokkermans, editors, *Automated Reasoning - 10th International Joint Conference, IJCAR 2020, Proceedings, Part II*, volume 12167 of *Lecture Notes in Computer Science*, pages 416–435. Springer, 2020. doi:10.1007/978-3-030-51054-1_27.

14 Nao Hirokawa and Aart Middeldorp. Dependency pairs revisited. In Vincent van Oostrom, editor, *Rewriting Techniques and Applications, 15th International Conference, RTA 2004, Aachen, Germany, June 3-5, 2004, Proceedings*, volume 3091 of *Lecture Notes in Computer Science*, pages 249–268. Springer, 2004. doi:10.1007/978-3-540-25979-4_18.

15 Cynthia Kop and Naoki Nishida. Term rewriting with logical constraints. In Pascal Fontaine, Christophe Ringeissen, and Renate A. Schmidt, editors, *Frontiers of Combining Systems - 9th International Symposium, FroCoS 2013, Nancy, France, September 18-20, 2013. Proceedings*, volume 8152 of *Lecture Notes in Computer Science*, pages 343–358. Springer, 2013. doi:10.1007/978-3-642-40885-4_24.

16 Cynthia Kop and Naoki Nishida. Constrained term rewriting tool. In Martin Davis, Ansgar Fehnker, Annabelle McIver, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 20th International Conference, LPAR-20 2015, Suva, Fiji, November 24-28, 2015, Proceedings*, volume 9450 of *Lecture Notes in Computer Science*, pages 549–557. Springer, 2015. doi:10.1007/978-3-662-48899-7_38.

17 Salvador Lucas. Context-sensitive Rewriting. *ACM Comput. Surv.*, 53(4):78:1–78:36, 2020. doi:10.1145/3397677.

18 Salvador Lucas. Using Well-Founded Relations for Proving Operational Termination. *J. Autom. Reasoning*, 64(2):167–195, 2020. doi:10.1007/s10817-019-09514-2.

19 Salvador Lucas. Applications and extensions of context-sensitive rewriting. *Journal of Logical and Algebraic Methods in Programming*, 121:100680, 2021. doi:10.1016/j.jlamp.2021.100680.

20 Salvador Lucas. Confluence of Conditional Rewriting Modulo. In Aniello Murano and Alexandra Silva, editors, *32nd EACSL Annual Conference on Computer Science Logic (CSL 2024)*, volume 288 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 37:1–37:21, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CSL.2024.37.

21 Salvador Lucas. Local confluence of conditional and generalized term rewriting systems. *Journal of Logical and Algebraic Methods in Programming*, 136:paper 100926, pages 1–23, 2024. doi:10.1016/j.jlamp.2023.100926.

22 Salvador Lucas, Claude Marché, and José Meseguer. Operational termination of conditional term rewriting systems. *Inf. Process. Lett.*, 95(4):446–453, 2005. doi:10.1016/j.ipl.2005.05.002.

23 Salvador Lucas and José Meseguer. Normal forms and normal theories in conditional rewriting. *J. Log. Algebr. Meth. Program.*, 85(1):67–97, 2016. doi:10.1016/j.jlamp.2015.06.001.

24 Salvador Lucas and José Meseguer. Dependency pairs for proving termination properties of conditional term rewriting systems. *J. Log. Algebr. Meth. Program.*, 86(1):236–268, 2017. doi:10.1016/j.jlamp.2016.03.003.

25 Salvador Lucas, José Meseguer, and Raúl Gutiérrez. The 2D Dependency Pair Framework for conditional rewrite systems. Part I: Definition and basic processors. *J. Comput. Syst. Sci.*, 96:74–106, 2018. doi:10.1016/j.jcss.2018.04.002.

26 Salvador Lucas, José Meseguer, and Raúl Gutiérrez. The 2D Dependency Pair Framework for Conditional Rewrite Systems. Part II: Advanced Processors and Implementation Techniques. *J. Autom. Reasoning*, 64(8):1611–1662, 2020. doi:10.1007/s10817-020-09542-3.

27 Elliott Mendelson. *Introduction to mathematical logic (4. ed.)*. Chapman and Hall, 1997.

28 Enno Ohlebusch. Termination of logic programs: Transformational methods revisited. *Appl. Algebra Eng. Commun. Comput.*, 12(1/2):73–116, 2001. doi:10.1007/s002000100064.

29 Enno Ohlebusch. *Advanced Topics in Term Rewriting*. Springer, 2002. doi:10.1007/978-1-4757-3661-8.

30 Dag Prawitz. *Natural deduction. A proof theoretical study*. Stockholm Studies in Philosophy. Almqvist & Wiksell, 1965.

31 John Alan Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *J. ACM*, 12(1):23–41, 1965. doi:10.1145/321250.321253.

32 Felix Schernhammer and Bernhard Gramlich. Characterizing and proving operational termination of deterministic conditional term rewriting systems. *J. Log. Algebraic Methods Program.*, 79(7):659–688, 2010. doi:10.1016/j.jlap.2009.08.001.

33 Terese. *Term rewriting systems*, volume 55 of *Cambridge tracts in theoretical computer science*. Cambridge University Press, 2003.

34 Yoshihito Toyama and Michio Oyamaguchi. Church-rosser property and unique normal form property of non-duplicating term rewriting systems. In Nachum Dershowitz and Naomi Lindenstrauss, editors, *Conditional and Typed Rewriting Systems, 4th International Workshop, CTRS-94, Jerusalem, Israel, July 13-15, 1994, Proceedings*, volume 968 of *Lecture Notes in Computer Science*, pages 316–331. Springer, 1994. doi:10.1007/3-540-60381-6_19.

State Canonization and Early Pruning in Width-Based Automated Theorem Proving

Mateus de Oliveira Oliveira
Department of Computer and Systems Sciences, Stockholm University, Sweden
Department of Informatics, University of Bergen, Norway

Farhad Vadiee
Department of Informatics, University of Bergen, Norway

Abstract

Width-based automated theorem proving is a framework where counter-examples for graph theoretic conjectures are searched width-wise relative to some graph width measure, such as treewidth or pathwidth. In a recent work it has been shown that dynamic programming algorithms operating on tree decompositions can be combined together with the purpose of width-based theorem proving. This approach can be used to show that several long-standing conjectures in graph theory can be tested in time $2^{2^{k^{O(1)}}}$ on the class of graphs of treewidth at most k. In this work, we give the first steps towards evaluating the viability of this framework from a practical standpoint. At the same time, we advance the framework in two directions. First, we introduce a state-canonization technique that significantly reduces the number of states evaluated during the search for a counter-example of the conjecture. Second, we introduce an early-pruning technique that can be applied in the study of conjectures of the form $\mathbb{P}_1 \rightarrow \mathbb{P}_2$, for graph properties $\mathbb{P}_1$ and $\mathbb{P}_2$, where $\mathbb{P}_1$ is a property closed under subgraphs.

As a concrete application, we use our framework in the study of graph theoretic conjectures related to coloring triangle free graphs. In particular, our algorithm is able to show that Reed's conjecture for triangle free graphs is valid on the class of graphs of pathwidth at most 5, and on graphs of treewidth at most 3. Perhaps more interestingly, our algorithm is able to construct in a completely automated way counter-examples for non-valid strengthenings of Reed's conjecture. These are the first results showing that width-based automated theorem proving is a promising avenue in the study of graph-theoretic conjectures.

2012 ACM Subject Classification Theory of computation → Automated reasoning; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Width-Based Automated Theorem Proving, Dynamic Programming, Parameterized Complexity

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.33

Funding Research Council of Norway, Project Number 288761

1 Introduction

Width-based automated theorem proving is a framework where parameterized algorithms are employed to search for counterexamples to graph-theoretic conjectures. Within this framework, the search for counterexamples is conducted width-wise, relative to some specific width measure for graphs, such as treewidth or pathwidth. More specifically, given a conjecture C and a positive integer k, the objective is to determine whether C holds on the class of graphs of width at most k. If C does not hold on this class of graphs, a counterexample of width at most k that invalidates the conjecture should be produced.

This approach is relevant for two main reasons. First, many interesting classes of graphs have small width with respect to some graph width measure. For example, trees and forests have treewidth at most 1, series-parallel graphs have treewidth at most 2 and outerplanar

graphs have treewidth at most 2 [16, 2, 5]. It has also been shown that k-outerplanar graphs have treewidth $O(k)$ [1, 15], and that k-caterpilars have pathwidth at most k [20]. Second, many important conjectures in graph theory are not known to hold on classes of graphs of small treewidth or pathwidth, and it is therefore natural to try to determine whether such conjectures hold when restricted to such classes of graphs. In many cases, structural properties of graphs of small treewidth or pathwidth have been used to produce analytic proofs of special cases of several well studied conjectures [11, 6, 12, 13, 19, 22, 13, 17]. The framework of width-based automated theorem proving is an avenue for automatizing this approach for certain classes of conjectures.

In a recent work [9], a new approach for width-based automated theorem proving was introduced. Within this approach, instead of specifying graph properties using logical formulas, such properties are specified using dynamic programming algorithms operating on graph decompositions. More specifically, it was shown in [9] that given dynamic programming cores $\mathsf{D}_1, \mathsf{D}_2, \ldots, \mathsf{D}_r$ specifying graph properties $\mathbb{P}_1, \mathbb{P}_2, \ldots, \mathbb{P}_r$, then for any Boolean combination $\mathbb{P}$ of these properties, there is an algorithm $\mathfrak{A}$ that takes a number k as input and decides whether all graphs of treewidth at most k belong to $\mathbb{P}$. Furthermore, if not all graphs of treewidth k belong to $\mathbb{P}$, then the algorithm outputs a certificate that can be used to extract a counter-example. This approach takes double-exponential time with respect to the number of bits needed to represent local witnesses used by the dynamic programming algorithms. Given that many interesting graph properties have DP-algorithms that use local witnesses of size $k^{O(1)}$ when processing tree decompositions of width k, the dynamic programming approach developed in [9] implies that many interesting conjectures can be tested in time double-exponential in $k^{O(1)}$ on the class of graphs of treewidth at most k. This includes long-standing conjectures such as Hadwiger's conjecture (for a fixed number of colors) [4], Tutte's flow conjectures [14, 18], Reed triangle free conjecture etc.

1.1 Our Results

In this work we give the first steps towards evaluating the viability of the width-based automated theorem proving framework introduced in [9] from a practical perspective. At the same time, we advance this framework by introducing a new width based deduction algorithm that produces significantly less states when compared with the algorithm originally introduced in [9]. Our new algorithm leverages on the introduction of two techniques. First, we introduce a suitable notion of state canonization. The second is an early pruning technique that can be used in the study of conjectures of the form $\mathbb{P}_1 \to \mathbb{P}_2$, where $\mathbb{P}_1$ is a graph property closed under subgraphs. As a concrete case study, we use our implementation to study graph-theoretic statements related to colorings of graphs. Our algorithm was able to produce non-trivial counter-examples for false statements, and also to confirm a well known conjecture due to Reed [21] on the class of graphs of pathwidth at most 5, and on the class of graphs of treewidth at most 3. Together, our results provide the first evidence that the width-based ATP framework introduced in [9] may be a promising avenue in the study of graph-theoretic conjectures.

1.2 Related Work

The approach for width-based automated theorem proving introduced in [9], and further developed in the present work, is heavily based on dynamic programming algorithms deciding graph theoretic properties. Another suitable approach for the study of width-based automated theorem proving is based on logic [7, 8]. In this context, instead of using dynamic-programming

algorithms to define graph properties, such graph properties are defined using logical formulas. For example, it can be shown that for each formula φ in the monadic second order logic of graphs, there is an algorithm that takes a positive integer k as input and determines whether every graph of treewidth at most k satisfies φ [7]. The drawback with this approach is that the running time of the deduction algorithm is governed by a function that grows as a tower of exponentials on the quantifier depth of the formula φ.

It is worth noting that several interesting graph theoretic properties have dynamic programming algorithms that produce local witnesses of size $k^{O(1)}$ when processing tree decompositions of width at most k, while at the same time, require large quantifier depth to be expressed in MSO logic [9]. For such properties, while the time complexity implied by the logic approach is upper bounded by a tower of exponentials on k, the time complexity implied by the dynamic-programming approach is upper bounded double exponential in $k^{O(1)}$.

2 Basic Definitions

Basic Notation. We let $\mathbb{N} \doteq \{0, 1, \dots\}$ denote the set of natural numbers and $\mathbb{N}_+ = \mathbb{N}\backslash\{0\}$ denote the set of positive natural numbers. Given a number $n \in \mathbb{N}$, we let $[n] \doteq \{1, \dots, n\}$. In particular, $[0] = \emptyset$. Given a set S, the set of finite subsets of S is denoted by $\mathcal{P}_{\text{fin}}(S)$, the set of all subsets of S is denoted by $\mathcal{P}(S)$.

Graphs. In this work, a *graph* is a triple $G = (V, E, \rho)$ where $V \subseteq \mathbb{N}$ is a *finite* set of *vertices*, $E \subseteq \mathbb{N}$ is a finite set *edges*, and $\rho \subseteq E \times V$ is an *incidence relation* with the property that each edge is incident to exactly two vertices. For each edge $e \in E$, we let $\text{endpts}(e) = \{v \in V \ : \ (e, v) \in \rho\}$ be the *endpoints* of e. In what follows, we may write $V(G)$, $E(G)$ and ρ_G to denote the sets V, E and ρ respectively. We let $|G| = |V(G)| + |E(G)|$ be the *size* of G. We let Graphs denote the set of all graphs. For us, the *emtpy graph* is the graph $(\emptyset, \emptyset, \emptyset)$ with no vertices and no edges.

Graph Isomorphisms. An *isomorphism* from a graph G to a graph H is a pair $\varphi = (\phi, \nu)$ where $\phi : V(G) \to V(H)$ is a bijection from the vertices of G to the vertices of H and $\nu : E(G) \to E(H)$ is a bijection from the edges of G to the edges of H with the property that for each vertex $v \in V(G)$ and each edge $e \in E(G)$, $(v, e) \in \rho_G$ if and only if $(\phi(v), \nu(e)) \in \rho_H$. If such a bijection exists, we say that G and H are *isomorphic*, and denote this fact by $G \sim H$.

Graph Properties. A *graph property* is any subset $\mathbb{P} \subseteq$ Graphs closed under isomorphisms. That is to say, for each two isomorphic graphs G and H in Graphs, $G \in \mathbb{P}$ if and only if $H \in \mathbb{P}$. Note that the sets $\emptyset$ and Graphs are graph properties. Given a set S of graphs, the *isomorphism closure* of S is defined as the set $\mathsf{ISO}(S) = \{G \in \text{Graphs} \ : \ \exists H \in S, G \sim H\}$.

Ranked Alphabet. A *ranked alphabet* is a finite set Σ together with function $\mathfrak{a} : \Sigma \to \mathbb{N}$, which intuitively specifies the arity of each symbol in Σ. A term over Σ is a pair $\tau = (T, \lambda)$ where T is a rooted tree and $\lambda : \mathsf{Nodes}(T) \to \Sigma$ is a function that labels each node p in $\mathsf{Nodes}(T)$ with a symbol from Σ in such a way that if $\lambda(p)$ is a symbol of arity r, then p has r children $p_1, \dots, p_r$. In particular, leaf nodes are labeled with symbols of arity 0. We assume that the children of p are ordered from left to right, so it makes sense to speak about th i-th child of a node. We may write $\mathsf{Nodes}(\tau)$ to refer to $\mathsf{Nodes}(T)$. We write $|\tau|$ to denote $|\mathsf{Nodes}(T)|$. The height of τ is defined as the height of T. We denote by $\text{Terms}(\Sigma)$ the set of all terms over Σ.

Terms. If $\tau_1 = (T_1, \lambda_1), ..., \tau_r = (T_r, \lambda_r)$ are terms in $\mathrm{Terms}(\Sigma)$, and $a \in \Sigma$ is a symbol of arity r, then we let $a(\tau_1, ..., \tau_r)$ denote the term $\tau = (T, \lambda)$ where $\mathsf{Nodes}(T) = \{u\} \cup \mathsf{Nodes}(T_1) \cup \ldots \cup \mathsf{Nodes}(T_r)$ for some fresh node u, $\mathsf{root}(T) = u$, $\lambda(u) = a$, and $\lambda|_{\mathsf{Nodes}(T_j)} = \lambda_j$ for each $j \in [r]$.

3 Instructive Dynamic Programming Cores

In this work, we represent graphs of treewidth at most k using k-instructive tree decompositions (k-ITDs) [9]. For each $k \in \mathbb{N}$, consider the following alphabet

$$\Sigma_k = \Big\{\texttt{Leaf},\ \texttt{IntroVertex}\{u\}, \texttt{IntroEdge}\{u,v\}, \texttt{ForgetVertex}\{u\},\ \texttt{Join} \ :\ u,v \in [k+1], u \neq v\Big\}, \tag{1}$$

where `Leaf` is a symbol of arity 0, $\texttt{IntroVertex}\{u\}$, $\texttt{ForgetVertex}\{u\}$ and $\texttt{IntroEdge}\{u,v\}$ are symbols of arity 1, and `Join` is a symbol of arity 2. We call Σ_k the *k-instructive alphabet*. Intuitively, elements of Σ_k represent instructions that can be used for the construction of a graph of treewidth at most k, together with a set $\mathfrak{b} \subseteq [k+1]$ of *active labels*, where each active label labels exacly one vertex of the graph.

1. In the base case, the instruction `Leaf` creates an empty graph with an empty set of active labels.
2. Now, let G be a graph with set of active labels $\mathfrak{b}$.
   a. For each $u \in [k+1] \setminus \mathfrak{b}$, the instruction $\texttt{IntroVertex}\{u\}$ adds a new vertex to G, labels this vertex with u, and adds u to $\mathfrak{b}$.
   b. For each $u \in \mathfrak{b}$, the instruction $\texttt{ForgetVertex}\{u\}$ erases the label from the current vertex labeled with u, and removes u from $\mathfrak{b}$. The intuition is that the label u is now free and may be used later in the creation of another vertex.
   c. For each $u, v \in \mathfrak{b}$, the instruction $\texttt{IntroEdge}\{u,v\}$ introduces a new edge between the current vertex labeled with u and the current vertex labeled with v. We note that multiedges are allowed in our graphs.
3. Finally, if G and G' are two graphs, each having $\mathfrak{b}$ as the set of active labels, then the instruction `Join` creates a new graph by identifying, for each $u \in \mathfrak{b}$, the vertex of G labeled with u with the vertex of G' labeled with u.

Such a construction process can be formalized using a term τ over Σ_k, which specifies an inductive construction from the leaves towards the root. More specifically, leaves are labeled with the `Leaf` instruction, nodes with a single child are labeled with an instruction of type $\texttt{IntroVertex}\{u\}$, $\texttt{ForgetVertex}\{u\}$, or $\texttt{IntroEdge}\{u,v\}$, and nodes with two children are labeled with the `Join` instruction. We let $\mathcal{G}(\tau)$ be the graph associated with the root of τ. We let $B(\tau)$ be the set of active labels after processing all operations in τ, and let $\theta[\tau] : B(\tau) \to V(\mathcal{G}(\tau))$ be the map that sends each label in $B(\tau)$ to its corresponding vertex in $\mathcal{G}(\tau)$.

We note that not all terms over Σ_k give rise to legal graphs. For instance, if the set $\mathfrak{b}$ does not contain a label u then the instruction $\texttt{ForgetVertex}\{u\}$ is not well defined. Similarly, if u is already in $\mathfrak{b}$, then the instruction $\texttt{IntroVertex}\{u\}$ is not well defined. In order to specify the set of all terms over Σ_k that do correspond to graphs, we may use a tree automaton A_k. More specifically, we let $A_k = (\Sigma_k, Q_k, F_k, \Delta_k)$ be the tree automaton where $Q_k = F_k = \mathcal{P}([k+1])$, and

$$\begin{aligned}\Delta_k &= \{\texttt{Leaf} \to \emptyset\}\\ &\cup \{\texttt{IntroVertex}\{u\}(\mathfrak{b}) \to \mathfrak{b} \cup \{u\} \mid u \notin [k+1] \backslash \mathfrak{b}\}\\ &\cup \{\texttt{ForgetVertex}\{u\}(\mathfrak{b}) \to \mathfrak{b} \backslash \{u\} \mid u \in \mathfrak{b}\}\\ &\cup \{\texttt{IntroEdge}\{u,v\}(\mathfrak{b}) \to \mathfrak{b} \mid u,v \in \mathfrak{b}, u \neq v\}\\ &\cup \{\texttt{Join}(\mathfrak{b},\mathfrak{b}') \to \mathfrak{b} \mid \mathfrak{b} = \mathfrak{b}'\}.\end{aligned}$$

We let $\mathsf{ITD}_k = \mathcal{L}(A_k)$ where $\mathcal{L}(A_k)$ is the set of terms accepted by A_k. The terms in ITD_k are called *k-instructive decompositions*. It turns out that graphs that can be represented by k-instructive tree decompositions are precisely the graphs of treewidth at most k.

▶ **Lemma 1** ([9]). *Let $G \in$ GRAPHS and $k \in \mathbb{N}$. Then G has treewidth at most k if and only if there exists a k-instructive tree decomposition τ such that $\mathcal{G}(\tau) \simeq G$.*

Dynamic programming algorithms operating on tree decompositions can be formalized using the notion of an *instructive dynamic programming core* (instructive DP-core), as defined below.

▶ **Definition 2** (Instructive DP-Cores). *An* instructive dynamic programming core *is a sequence of 6-tuples $\mathsf{D} = \{(\Sigma_k, \mathcal{W}_k, \texttt{Final}_k, \Delta_k, \texttt{Clean}_k, \texttt{Inv}_k)\}_{k\in\mathbb{N}}$ where for each $k \in \mathbb{N}$,*

1. *Σ_k is the k-instructive alphabet;*
2. *$\mathcal{W}_k \subseteq \{0,1\}^*$ is a decidable subset of $\{0,1\}^*$;*
3. *$\texttt{Final}_k : \mathcal{W}_k \to \{0,1\}$ is a function;*
4. *Δ_k is a set containing*
   - *A* finite *subset $\texttt{Leaf} \subseteq \mathcal{W}$.*
   - *A function $\texttt{IntroVertex}\{u\} : \mathcal{W} \to \mathcal{P}_{\text{fin}}(\mathcal{W})$ for each $u \in [k+1]$.*
   - *A function $\texttt{ForgetVertex}\{u\} : \mathcal{W} \to \mathcal{P}_{\text{fin}}(\mathcal{W})$ for each $u \in [k+1]$.*
   - *A function $\texttt{IntroEdge}\{u,v\} : \mathcal{W} \to \mathcal{P}_{\text{fin}}(\mathcal{W})$ for each $\{u,v\} \in \mathcal{P}([k+1], 2)$.*
   - *A function $\texttt{Join} : \mathcal{W} \times \mathcal{W} \to \mathcal{P}_{\text{fin}}(\mathcal{W})$.*
5. *$\texttt{Clean}_k : \mathcal{P}_{\text{fin}}(\mathcal{W}) \to \mathcal{P}_{\text{fin}}(\mathcal{W})$ is a function;*
6. *$\texttt{Inv}_k : \mathcal{P}_{\text{fin}}(\mathcal{W}) \to \{0,1\}^*$ is a function.*

For each $k \in \mathbb{N}$, we let $\mathsf{D}[k] = (\Sigma_k, \mathcal{W}_k, \texttt{Final}_k, \Delta_k, \texttt{Clean}_k, \texttt{Inv}_k)$ denote the k-th tuple of D. We may write $\mathsf{D}[k].\Sigma$ to denote the set Σ_k, $\mathsf{D}[k].\mathcal{W}$ to denote the set $\mathcal{W}_k$, and so on. Intuitively, for each k, $\mathsf{D}[k]$ is a description of a dynamic programming algorithm that operates on k-instructive tree decompositions. Such an algorithm processes a k-instructive tree decomposition τ from the leaves towards the root, and assigns a set of local witnesses to each node of τ, depending on which instruction labels the node and on the sets assigned to the children of the node. Some dynamic programming algorithms use a function that removes redundant local witnesses from the set of local witnesses constructed at each node. In our framework, this is formalized by the function $\texttt{Clean}_k$. In this work, we assume that $\texttt{Clean}_k$ is the identity function. Finally, the function $\texttt{Inv}_k$ is used whenever we want to use dynamic programming algorithms to compute graph invariants. In this work, we will not be concerned with the computation of invariants, and therefore, we assume that $\texttt{Inv}_k$ is the Boolean function that assigns 1 to a set of local witnesses if and only if it contains some final witness. This process is formalized by the notion of *dynamization*, which we define below.

▶ **Definition 3** (Dynamization). *Let $k \in \mathbb{N}$ and D be an instructive DP-core. The k-dynamization of D is the function $\Gamma[\mathsf{D}, k] : \mathsf{ITD}_k \to \mathcal{P}_{\text{fin}}(\mathsf{D}[k].\mathcal{W})$ inductively defined as follows for each $\tau \in \mathsf{ITD}_k$.*

1. *If $\tau = \texttt{Leaf}$, then $\Gamma[\mathsf{D}, k](\tau) = \mathsf{D}[k].\texttt{Leaf}$.*

2. *If* $\tau = \texttt{IntroVertex}\{u\}(\sigma)$*, then* $\Gamma[\mathsf{D}, k](\tau) = \mathsf{D}[k].\texttt{Clean}(\texttt{IntroVertex}\{u\}(\Gamma[\mathsf{D}, k](\sigma)))$.
3. *If* $\tau = \texttt{ForgetVertex}\{u\}(\sigma)$*, then* $\Gamma[\mathsf{D}, k](\tau) = \mathsf{D}[k].\texttt{Clean}(\texttt{ForgetVertex}\{u\}(\Gamma[\mathsf{D}, k](\sigma)))$.
4. *If* $\tau = \texttt{IntroEdge}\{u, v\}(\sigma)$*, then* $\Gamma[\mathsf{D}, k](\tau) = \mathsf{D}[k].\texttt{Clean}(\texttt{IntroEdge}\{u, v\}(\Gamma[\mathsf{D}, k](\sigma)))$.
5. *If* $\tau = \texttt{Join}(\sigma_1, \sigma_2)$*, then* $\Gamma[\mathsf{D}, k](\tau) = \mathsf{D}[k].\texttt{Clean}(\texttt{Join}(\Gamma[\mathsf{D}, k](\sigma_1), \Gamma[\mathsf{D}, k](\sigma_2)))$.

We say that $\mathsf{D}[k]$ accepts τ if $\Gamma[\mathsf{D}, k](\tau)$ has a final local witness, i.e. a local witness w with the property that $\mathsf{D}[k].\texttt{Final}(\mathsf{w}) = 1$. We let $\mathbb{G}(\mathsf{D}[k]) = \mathsf{ISO}(\{\mathcal{G}(\tau) : \tau \text{ is accepted by } \mathsf{D}[k]\})$ be the isomorphism closure of the set of graphs associated with terms accepted by $\mathsf{D}[k]$. We note that $\mathbb{G}(\mathsf{D}[k])$ is a graph property, and that all graphs in $\mathbb{G}(\mathsf{D}[k])$ have treewidth at most k. We let $\mathbb{G}(\mathsf{D}) = \bigcup_{k \in \mathbb{N}} \mathbb{G}(\mathsf{D}[k])$ be the graph property defined by D.

▶ **Definition 4** (Coherency). *Let* D *be an instructive DP-core. We say that* D *is coherent if for each* $k, k' \in \mathbb{N}$*, each* $\tau \in \mathsf{ITD}_k$*, and each* $\tau' \in \mathsf{ITD}_{k'}$ *if* $\mathcal{G}(\tau) \simeq \mathcal{G}(\tau')$ *then* $\mathsf{D}[k]$ *accepts* τ *if and only if* $\mathsf{D}[k']$ *accepts* τ'*.*

Let D be a coherent instructive DP-core, and $k \in \mathbb{N}$. A (k, D)-state is a pair of the form $(\mathfrak{b}, S)$ where $\mathfrak{b} \subseteq [k+1]$ and $S \subseteq \mathsf{D}.\mathcal{W}$. Such a state is said to be (k, D)-*inconsistent*, if S has no final local witness. The initial (k, D)-state is the pair $(\emptyset, \mathsf{D}.\texttt{Leaf})$.

▶ **Definition 5** ((k, D)-Refutation). *Let* D *be an instructive DP-core. A* (k, D)*-refutation is a sequence of pairs* $R \equiv (\mathfrak{b}_0, S_0)(\mathfrak{b}_1, S_1) \dots (\mathfrak{b}_m, S_m)$ *satisfying the following conditions.*

1. $(\mathfrak{b}_0, S_0) = (\emptyset, \mathsf{D}[k].\texttt{Leaf})$.
2. $(\mathfrak{b}_m, S_m)$ *is* (k, D)*-inconsistent.*
3. *For each* $i \in [m]$*, there is some* $j \in [i]$*, such that* $(\mathfrak{b}_i, S_i)$ *is equal to one of the following pairs.*
   a. $(\mathfrak{b}_j \cup \{u\}, \texttt{IntroVertex}\{u\}(S_j))$ *with* $u \notin \mathfrak{b}_j$.
   b. $(\mathfrak{b}_j \setminus \{u\}, \texttt{ForgetVertex}\{u\}(S_j))$ *with* $u \in \mathfrak{b}_j$.
   c. $(\mathfrak{b}_j, \texttt{IntroEdge}\{u, v\}(S_j))$ *with* $u, v \in \mathfrak{b}_j$.
   d. $(\mathfrak{b}_j, \texttt{Join}(S_j, S_l))$ *with* $l \in [i]$.

Intuitively, a (k, D)-refutation is a certificate that some inconsistent (k, D)-state is reachable from the initial (k, D)-state. It turns out that if D is a coherent instructive DP-core, then constructing a (k, D)-refutation is equivalent to showing that $\mathbb{G}(\mathsf{D})$ does not contain all graphs of treewidth at most k.

▶ **Theorem 6** ([9]). *Let* D *be a coherent instructive DP-core. Then there is a* (k, D)*-refutation if and only if some graph of treewidth at most* k *does not belong to the graph property* $\mathbb{G}(\mathsf{D})$*.*

4 Example: An Instructive DP-Core for Chromatic Number at Most r

Let S be a finite set, and $r \in \mathbb{N}$. An r-partition of S is a partition of S with at most r cells. Let G be a graph. We say that G is r-colorable if there is an r-partition of $V(G)$ such that for each edge $e \in E(G)$, the endpoints of e belong to distinct cells. Let $\textsc{Colorable}_r$ be the graph property consisting of all graphs that are r-colorable. In this section, we specify a DP-core C-$\textsc{Colorable}_r$ for the property $\textsc{Colorable}_r$. We start by defining the notion of a C-$\textsc{Colorable}_r[k]$ local witness, for each $k \in \mathbb{N}$.

▶ **Definition 7.** *Let* $k \in \mathbb{N}$*. A C-*$\textsc{Colorable}_r[k]$ *local witness is any* r*-partition of a subset of* $[k+1]$*.*

▶ **Definition 8.** *Let $r \in \mathbb{N}$. We let C-Colorable$_r$ be the instructive DP-core* D *specified below. For each $k \in \mathbb{N}$, we define C-Colorable$_r[k] = \mathsf{D}[k]$. We let $u, v \in [k+1]$,* w *and* w' *be C-Colorable$_r[k]$ local witnesses, and S be a set of such local witnesses.*

1. $\mathsf{D}[k].\mathcal{W} = \{\mathsf{w} : \mathsf{w}$ *is a C-Colorable$_r[k]$ local witness* $\}$.
2. $\mathsf{D}[k].\texttt{Leaf} = \{\emptyset\}$.
3. $\mathsf{D}[k].\texttt{IntroVertex}\{u\}(\mathsf{w}) = \begin{cases} \{(\mathsf{w} \setminus \{p\}) \cup \{p \cup \{u\}\} : p \in \mathsf{w}\} & \text{if } |\mathsf{w}| = r, \\ \{\mathsf{w} \cup \{\{u\}\}\} \cup \{(\mathsf{w} \setminus \{p\}) \cup \{p \cup \{u\}\} : p \in \mathsf{w}\} & \text{if } |\mathsf{w}| < r. \end{cases}$
4. $\mathsf{D}[k].\texttt{ForgetVertex}\{u\}(\mathsf{w}) = \{p \setminus \{u\} : p \in \mathsf{w}\} \setminus \{\emptyset\}$ [1].
5. $\mathsf{D}[k].\texttt{IntroEdge}\{u, v\}(\mathsf{w}) = \begin{cases} \{\mathsf{w}\} & \text{if } u \text{ and } v \text{ are not in a same cell,} \\ \emptyset & \text{Otherwise.} \end{cases}$
6. $\mathsf{D}[k].\texttt{Join}(\mathsf{w}, \mathsf{w}') = \begin{cases} \{\mathsf{w}\} & \text{if } \mathsf{w} = \mathsf{w}', \\ \emptyset & \text{Otherwise.} \end{cases}$
7. $\mathsf{D}[k].\texttt{Final}(\mathsf{w}) = 1$ *for every* $\mathsf{w} \in \mathsf{D}.\mathcal{W}$.
8. $\mathsf{D}[k].\texttt{Clean}(S) = S$ *for every* $S \subseteq \mathsf{D}.\mathcal{W}$.
9. $\mathsf{D}[k].\texttt{Inv}(S) = \begin{cases} 1 & \text{if } S \text{ has a final witness,} \\ 0 & \text{Otherwise.} \end{cases}$

Next, we define a predicate relating k-instructive tree decompositions with local witnesses.

▶ **Definition 9.** *We let P-Colorable$_r[k]$ be the predicate that is true on a pair $(\tau, \mathsf{w}) \in \mathsf{ITD}_k \times$ C-Colorable$_r[k].\mathcal{W}$ if and only if the following conditions are satisfied.*

1. $\bigcup_{c \in \mathsf{w}} c = dom(\theta[\tau])$.
2. *There is an r-partition α of $V(\mathcal{G}(\tau))$ such that for every $u, v \in B(\tau)$, $\theta[\tau](u)$ and $\theta[\tau](v)$ belong to the same cell in α if and only if u and v belong to the same cell in* w.

▶ **Proposition 10.** *For each $\tau \in \mathsf{ITD}_k$, a local witness* w *belongs to Γ[C-Colorable$_r, k](\tau)$ if and only if P-Colorable$_r[k](\tau, \mathsf{w}) = 1$.*

The next corollary states that the predicate P-Colorable$[k, r]$ characterizes those pairs (τ, w) for which w is a witness in Γ[C-Colorable$_r[k], \tau]$. Below, Accepted(C-Colorable$_r[k]$) is the set of k-instructive tree decompositions accepted by C-Colorable$_r[k]$.

▶ **Corollary 11.** *Let τ be a k-instructive tree decomposition. Then $\mathcal{G}(\tau)$ is r-colorable if and only if $\tau \in$* Accepted*(C-Colorable$_r[k]$).*

Since a C-Colorable$_r[k]$ local witness is an r-partition of a subset of $[k+1]$, we can represent such a partition using $O(k \cdot \log r)$ bits.

▶ **Observation 12.** *C-Colorable$_r[k]$ has bit-length $O(k \cdot \log r)$.*

[1] There is at most one cell containing u. If this cell is a singleton, the whole cell is deleted from w.

5 Width Based ATP with Symmetry Breaking

In this section, we introduce our main technical result. More specifically, we introduce a width-based automated deduction algorithm endowed with a symmetry breaking procedure that allows us to remove redundant states during the search for counter-example for a given conjecture. At the core of our technique, lies the notion of a *witness action*. Intuitively, functions that satisfy the axioms of a witness action can be used to define permuted versions of local witnesses generated by a DP-core. This allows us to define the notion of canonical form of a state generated during the search process. Instead of keeping track of all inferred states, we only keep their canonical forms. This leads to a significant reduction of the search space because states with the same canonical form are identified. Our main theorem (Theorem 19) states that this process preserves provability.

Let $\mathcal{F}_k = \{f : \mathfrak{b} \to [k+1] \mid \mathfrak{b} \subseteq [k+1]\}$ be the set of all injective functions $f : \mathfrak{b} \to [k+1]$ from some subset $\mathfrak{b} \subseteq [k+1]$ to $[k+1]$. We call the elements of $\mathcal{F}$ *relabeling functions*. Given such a function $f \in \mathcal{F}_k$, and a subset $\mathfrak{b} \subseteq [k+1]$, we let $f(\mathfrak{b}) = \{f(u) \ : \ u \in \mathfrak{b}\}$ be the image of $\mathfrak{b}$ under f. Next, we introduce the notion of a *witness action* for a DP-core (Definition 13). Actions will be used later to define the notion of a canonical form for a $\mathsf{D}[k]$-state.

▶ **Definition 13** (Witness Action). *Let* D *be a DP-core and* $k \in \mathbb{N}$. *We say that a function* $\rho^k_{\mathsf{D}} : \mathcal{F}_k \times \mathsf{D}[k].\mathcal{W} \to \mathsf{D}[k].\mathcal{W}$ *is an action for* $\mathsf{D}[k]$ *if the following conditions are satisfied for each* $f \in \mathcal{F}_k$, *and each* $\mathsf{w} \in \mathsf{D}[k].\mathcal{W}$.

1. ρ^k_{D} *preserves acceptance:* $\mathsf{w} \in \mathsf{Accepted}(\mathsf{D}[k])$ *if and only if* $\rho^k_{\mathsf{D}}(f, \mathsf{w}) \in \mathsf{Accepted}(\mathsf{D}[k])$.
2. $\rho^k_{\mathsf{D}}(f^{-1}, \rho^k_{\mathsf{D}}(f, \mathsf{w})) = \mathsf{w}$.
3. $\rho^k_{\mathsf{D}}(f \circ f', \mathsf{w}) = \rho^k_{\mathsf{D}}(f, \rho^k_{\mathsf{D}}(f', \mathsf{w}))$.
4. $\rho^k_{\mathsf{D}}(f, \mathtt{IntroVertex}\{u\}(\mathsf{w})) = \mathtt{IntroVertex}\{f(u)\}(\rho^k_{\mathsf{D}}(f, \mathsf{w}))$.
5. $\rho^k_{\mathsf{D}}(f, \mathtt{ForgetVertex}\{u\}(\mathsf{w})) = \mathtt{ForgetVertex}\{f(u)\}(\rho^k_{\mathsf{D}}(f, \mathsf{w}))$.
6. $\rho^k_{\mathsf{D}}(f, \mathtt{IntroEdge}\{u, v\}(\mathsf{w})) = \mathtt{IntroEdge}\{f(u), f(v)\}(\rho^k_{\mathsf{D}}(f, \mathsf{w}))$.
7. $\rho^k_{\mathsf{D}}(f, \mathtt{Join}(\mathsf{w}_1, \mathsf{w}_2)) = \mathtt{Join}(\rho^k_{\mathsf{D}}(f, \mathsf{w}_1), \rho^k_{\mathsf{D}}(f, \mathsf{w}_2))$.

We extend Definition 13 to subsets of witnesses by setting

$$\rho^k_{\mathsf{D}}(f, S) = \{\rho^k_{\mathsf{D}}(f, \mathsf{w}) \ : \ \mathsf{w} \in S\}$$

for each $S \subseteq \mathsf{D}[k].\mathcal{W}$. Next, given a DP-core D, we will define a notion of canonization for a pair $(\mathfrak{b}, S)$ where $\mathfrak{b} \subseteq [k+1]$ and $S \subseteq \mathcal{P}_{\text{fin}}(\mathsf{D}.\mathcal{W})$.

Let U be an ordered set of elements, and $X \subseteq U$. We let $\text{vec}(X)$ be the vector obtained by ordering the elements of X from the smallest value to the largest value. For instance, if $X = \{2, 3, 5\}$, then $\text{vec}(X) = (2, 3, 5)$. Given two such subsets $X, X' \subseteq U$, we say that $X < X'$ if $\text{vec}(X)$ is lexicographically smaller than $\text{vec}(X)$.

Let D be a DP-core and fix an arbitrary order for the set $\mathsf{D}[k].\mathcal{W}$. For instance, this order can be simply the lexicographic order on strings. We say that a pair $(\mathfrak{b}, S)$ is smaller than $(\mathfrak{b}', S')$ if the pair $(\text{vec}(\mathfrak{b}), \text{vec}(S))$ is lexicographically smaller than the pair $(\text{vec}(\mathfrak{b}'), \text{vec}(S'))$.

▶ **Definition 14** (Canonical Pair). *Let* $\mathfrak{b} \subseteq [k+1]$, $S \subseteq \mathsf{D}[k].\mathcal{W}$, *and*

$$\text{CAN}^k_{\mathsf{D}}(\mathfrak{b}, S) = \min\{(f(\mathfrak{b}), \rho^k_{\mathsf{D}}(f, S)) \mid f \in \mathcal{F}_k, \mathit{dom}(f) = \mathfrak{b}\}.$$

We call the function $f : \mathfrak{b} \to [k+1]$ *where the minimum in the above equation is achieved the* canonical relabeling of $(\mathfrak{b}, S)$.

Intuitively, given a fixed action ρ^k_{D} the canonical form $\mathrm{CAN}^k_{\mathsf{D}}(\mathfrak{b}, S)$ of a pair $(\mathfrak{b}, S)$ is the lexicographically smallest pair obtained by relabeling the elements of $\mathfrak{b}$ and each string $\mathsf{w} \in S$ according to some relabeling function $f : \mathfrak{b} \to [k+1]$. The canonical relabeling of $(\mathfrak{b}, q)$ is the unique function $f : \mathfrak{b} \to [k+1]$ with the property that $(f(\mathfrak{b}), \rho^k_{\mathsf{D}}(f, S)) = \mathrm{CAN}^k_{\mathsf{D}}(\mathfrak{b}, S)$.

Our next step is to define the notion of a relabeled refutation. Intuitively, relabelings will be used to produce refutations where all states are in canonical form.

▶ **Definition 15** (F-Relabeled Refutation). *Let $F = (f_1, \ldots, f_m)$ be a sequence of relabeling functions in $\mathcal{F}_k$ and D be a DP-core. An F-relabeled (k, D)-refutation is a sequence of pairs*

$$R \equiv (\mathfrak{b}_0, S_0)(\mathfrak{b}_1, S_1) \ldots (\mathfrak{b}_m, S_m)$$

satisfying the following conditions:

1. $(\mathfrak{b}_0, S_0) = (\emptyset, \mathsf{D}[k].\mathtt{Leaf})$.
2. $(\mathfrak{b}_m, S_m)$ *is (k, D)-inconsistent, i.e., S_m has no final local witness.*
3. *For each $i \in [m]$, there is some $j \in [i]$, such that $(\mathfrak{b}_i, S_i)$ is equal to one of the following pairs.*
   a. $(f_i(\mathfrak{b}_j \cup \{u\}), \rho^k_{\mathsf{D}}(f_i, \mathtt{IntroVertex}\{u\}(S_j)))$ *for some $u \notin \mathfrak{b}_j$.*
   b. $(f_i(\mathfrak{b}_j \setminus \{u\}), \rho^k_{\mathsf{D}}(f_i, \mathtt{ForgetVertex}\{u\}(S_j)))$ *for some $u \in \mathfrak{b}_j$.*
   c. $(f_i(\mathfrak{b}_j), \rho^k_{\mathsf{D}}(f_i, \mathtt{IntroEdge}\{u, v\}(S_j)))$ *for some $u, v \in \mathfrak{b}_j$.*
   d. $(f_i(\mathfrak{b}_j), \rho^k_{\mathsf{D}}(f_i, \mathtt{Join}(S_j, \rho^k_{\mathsf{D}}(f, S_l))))$ *for some $l \in [i]$ and some $f : \mathfrak{b}_l \to [k+1]$ with $f(\mathfrak{b}_l) = f_i(\mathfrak{b}_j)$.*

If the sequence R in Definition 15 satisfies Conditions 1 and 3, but not Condition 2, then we say that R is a semi F-relabeled (k, D)-refutation.

Below, given an instructive tree decomposition τ, we let $\mathrm{Sub}(\tau)$ denote the set of all subterms of τ.

▶ **Lemma 16.** *Let D be a DP-core, $F = (f_1, \ldots, f_m)$ be a sequence of k-relabeling functions. Let $R = (\mathfrak{b}_0, S_0)(\mathfrak{b}_1, S_1) \ldots (\mathfrak{b}_m, S_m)$ be a semi F-relabeled (k, D)-refutation, and $g : \mathfrak{b}_m \to [k+1]$ be a k-relabeling. Then, there is a k-instructive tree decomposition $\tau_R \in \mathsf{ITD}_k$ and a function $T : \mathrm{Sub}(\tau_R) \to \mathcal{P}_{\mathrm{fin}}(\mathsf{D}.\mathcal{W})$ such that the following conditions are satisfied for each subterm τ of τ_R.*

1. *If $\tau = \tau_R$, then $T(\tau) = \rho^k_{\mathsf{D}}(g, S_m)$.*
2. *if $\tau = \mathtt{Leaf}$, then $T(\tau) = \mathsf{D}[k].\mathtt{Leaf}$.*
3. *if $\tau = \mathtt{IntroVertex}\{u\}(\tau_1)$ for some subterm τ_1, then $T(\tau) = \mathtt{IntroVertex}\{u\}(T(\tau_1))$.*
4. *if $\tau = \mathtt{ForgetVertex}\{u\}(\tau_1)$ for some subterm τ_1, then $T(\tau) = \mathtt{ForgetVertex}\{u\}(T(\tau_1))$*
5. *if $\tau = \mathtt{IntroEdge}\{u, v\}(\tau_1)$ for some subterm τ_1, then $T(\tau) = \mathtt{IntroEdge}\{u, v\}(T(\tau_1))$.*
6. *if $\tau = \mathtt{Join}(\tau_1, \tau_2)$ for some subterms τ_1 and τ_2, then $T(\tau) = \mathtt{Join}(T(\tau_1), T(\tau_2))$.*

Proof. The proof of Lemma 16 follows by induction on m. In the base case, we have that $R = (\mathfrak{b}_0, S_0)$. In this case, by setting $\tau_R = \mathtt{Leaf}$ and $T(\tau_R) = \mathsf{D}[k].\mathtt{Leaf}$, the conditions 1-6 of Lemma 16 are satisfied.

Assume that the lemma holds for all $n < m$. We show that the lemma holds for $n = m$. Below, for each $i \in [n]$, we let $R_i = (\mathfrak{b}_0, S_0)(\mathfrak{b}_1, S_1) \ldots (\mathfrak{b}_i, S_i)$. By the induction hypothesis, for each such i, and each $h : \mathfrak{b}_i \to [k+1]$, there is a k-instructional tree decomposition τ^h_i and a function $T^h_i : \mathrm{Sub}(\tau') \to \mathcal{P}_{\mathrm{fin}}(\mathsf{D}.\mathcal{W})$ such that the conditions 1-6 of Lemma 16 are satisfied. There are four cases to be analyzed.

1. In the first case $(\mathfrak{b}_m, S_m) = (f_m(\mathfrak{b}_i \cup \{u\}), \rho^k_{\mathsf{D}}(f_m, \texttt{IntroVertex}\{u\}(S_i)))$ for some $i < m$. Let $\tau = \texttt{IntroVertex}\{g \circ f_m(u)\}(\tau_i^{g \circ f_m})$, and $T : \mathrm{Sub}(\tau) \to \mathcal{P}_{\mathrm{fin}}(\mathsf{D}[k].\mathcal{W})$ be such that $T(\tau) = \texttt{IntroVertex}\{g \circ f_m(u)\}(\rho^k_{\mathsf{D}}(g \circ f_m, S_i))$ and $T|_{\mathrm{Sub}(\tau_i^{g \circ f_m})} = T_i^{g \circ f_m}$. It should be clear that conditions 2-6 are satisfied. Finally, to show that Condition 1 is satisfied, we note that

$$\begin{aligned}
&\texttt{IntroVertex}\{g \circ f_m(u)\}(\rho^k_{\mathsf{D}}(g \circ f_m, S_i)) \\
&\quad = \rho^k_{\mathsf{D}}(g \circ f_m, \texttt{IntroVertex}\{u\}(S_i)) \\
&\quad = \rho^k_{\mathsf{D}}(g, \rho^k_{\mathsf{D}}(f_m, \texttt{IntroVertex}\{u\}(S_i))) \\
&\quad = \rho^k_{\mathsf{D}}(g, S_m).
\end{aligned}$$

   where the first equality follows from Definition 13.4, the second equality follows from Definition 13.3 and the third equality follows from the fact that $S_m = \rho^k_{\mathsf{D}}(f_m, \texttt{IntroVertex}\{u\}(S_i))$.
2. In the second case $(\mathfrak{b}_m, S_m) = (f_m(\mathfrak{b}_i \setminus \{u\}), \rho^k_{\mathsf{D}}(f_m, \texttt{ForgetVertex}\{u\}(S_i)))$ for some $i < m$. Let $\tau = \texttt{ForgetVertex}\{g \circ f_m(u)\}(\tau_i^{g \circ f_m})$, and $T : \mathrm{Sub}(\tau) \to \mathcal{P}_{\mathrm{fin}}(\mathsf{D}[k].\mathcal{W})$ be such that $T(\tau) = \texttt{ForgetVertex}\{g \circ f_m(u)\}(\rho^k_{\mathsf{D}}(g \circ f_m, S_i))$ and $T|_{\mathrm{Sub}(\tau_i^{g \circ f_m})} = T_i^{g \circ f_m}$. It should be clear that conditions 2-6 are satisfied. Finally, to show that Condition 1 is satisfied, we note that

$$\begin{aligned}
&\texttt{ForgetVertex}\{g \circ f_m(u)\}(\rho^k_{\mathsf{D}}(g \circ f_m, S_i)) \\
&\quad = \rho^k_{\mathsf{D}}(g \circ f_m, \texttt{ForgetVertex}\{u\}(S_i)) \\
&\quad = \rho^k_{\mathsf{D}}(g, \rho^k_{\mathsf{D}}(f_m, \texttt{ForgetVertex}\{u\}(S_i))) \\
&\quad = \rho^k_{\mathsf{D}}(g, S_m).
\end{aligned}$$

   where the first equality follows from Definition 13.5, the second equality follows from Definition 13.3 and the third equality follows from the fact that $S_m = \rho^k_{\mathsf{D}}(f_m, \texttt{ForgetVertex}\{u\}(S_i))$.
3. In the third case $(\mathfrak{b}_m, S_m) = (f_m(\mathfrak{b}_i), \rho^k_{\mathsf{D}}(f_m, \texttt{IntroEdge}\{u, v\}(S_i)))$ for some $i < m$. Let $\tau = \texttt{IntroEdge}\{g \circ f_m(u), g \circ f_m(v)\}(\tau_i^{g \circ f_m})$, and $T : \mathrm{Sub}(\tau) \to \mathcal{P}_{\mathrm{fin}}(\mathsf{D}[k].\mathcal{W})$ be such that $T(\tau) = \texttt{IntroEdge}\{g \circ f_m(u), g \circ f_m(v)\}(\rho^k_{\mathsf{D}}(g \circ f_m, S_i))$ and $T|_{\mathrm{Sub}(\tau_i^{g \circ f_m})} = T_i^{g \circ f_m}$. It should be clear that conditions 2-6 are satisfied. Finally, to show that Condition 1 is satisfied, we note that

$$\begin{aligned}
&\texttt{IntroEdge}\{g \circ f_m(u), g \circ f_m(v)\}(\rho^k_{\mathsf{D}}(g \circ f_m, S_i)) \\
&\quad = \rho^k_{\mathsf{D}}(g \circ f_m, \texttt{IntroEdge}\{u, v\}(S_i)) \\
&\quad = \rho^k_{\mathsf{D}}(g, \rho^k_{\mathsf{D}}(f_m, \texttt{IntroEdge}\{u, v\}(S_i))) \\
&\quad = \rho^k_{\mathsf{D}}(g, S_m).
\end{aligned}$$

   where the first equality follows from Definition 13.6, the second equality follows from Definition 13.3 and the third equality follows from the fact that $S_m = \rho^k_{\mathsf{D}}(f_m, \texttt{IntroEdge}\{u, v\}(S_i))$.
4. In the fourth case $(\mathfrak{b}_m, S_m) = (f_m(\mathfrak{b}_i), \rho^k_{\mathsf{D}}(f_m, \texttt{Join}(S_i, S_j)))$ for some $j, i < m$. Let $\tau = \texttt{Join}(\tau_i^{g \circ f_m}, \tau_j^{g \circ f_m})$, and $T : \mathrm{Sub}(\tau) \to \mathcal{P}_{\mathrm{fin}}(\mathsf{D}[k].\mathcal{W})$ be such that $T(\tau) = \texttt{Join}(\rho^k_{\mathsf{D}}(g \circ f_m, S_i), \rho^k_{\mathsf{D}}(g \circ f_m, S_j)$ and $T|_{\mathrm{Sub}(\tau_i^{g \circ f_m})} = T_i^{g \circ f_m}$ and $T|_{\mathrm{Sub}(\tau_j^{g \circ f_m})} = T_j^{g \circ f_m}$. It should be clear that conditions 2-6 are satisfied. Finally, to show that Condition 1 is satisfied, we note that

$$\begin{aligned}
&\texttt{Join}(\rho^k_{\mathsf{D}}(g \circ f_m, S_i), \rho^k_{\mathsf{D}}(g \circ f_m, S_j)) \\
&\quad = \rho^k_{\mathsf{D}}(g \circ f_m, \texttt{Join}(S_i, S_j)) \\
&\quad = \rho^k_{\mathsf{D}}(g, \rho^k_{\mathsf{D}}(f_m, \texttt{Join}(S_i, S_j))) \\
&\quad = \rho^k_{\mathsf{D}}(g, S_m).
\end{aligned}$$

   where the first equality follows from Definition 13.7, the second equality follows from Definition 13.3 and the third equality follows from the fact that $S_m = \rho^k_{\mathsf{D}}(f_m, \texttt{Join}(S_i, S_j))$.

◀

The next theorem states that if D is a coherent DP-core, then from each relabeled (k, D)-refutation, one can extract a k-instructive tree decomposition whose graph does not belong to $\mathbb{G}(\mathsf{D})$.

▶ **Theorem 17.** *Let $F = (f_1, \ldots, f_m)$ be a sequence of relabeling functions and D be a coherent DP-core. If there is an F-relabeled (k, D)-refutation, then there exists a k-instructive tree decomposition $\tau \in \mathsf{ITD}_k$ where $\mathcal{G}(\tau) \notin \mathbb{G}(\mathsf{D})$.*

Proof. Let $R = (\mathfrak{b}_0, S_0)(\mathfrak{b}_1, S_1) \ldots (\mathfrak{b}_m, S_m)$ be a F-relabeled (k, D)-refutation. By Lemma 16, there is a k-instructive tree decomposition τ and a map $T : \mathrm{Sub}(\tau) \to \mathcal{P}_{\mathrm{fin}}(\mathsf{D}.\mathcal{W})$ such that conditions 1-6 of Lemma 16 are satisfied. Conditions 2-6 imply that the value of T on τ is equal to the value of the dynamization of D on τ. In particular we have that $S_m = \Gamma[\mathsf{D}, k](\tau)$. Since R is a (k, D)-refutation, we have that S_m has no final local witness. Therefore, the definition of acceptance for a DP-core, τ is not accepted by D. Since D is coherent, then $\mathcal{G}(\tau)$ does not belong to $\mathbb{G}(\mathsf{D})$. ◀

Next, we define the notion of a canonized refutation. Intuitively, such a refutation is obtained by canonizing the (k, D)-pairs occurring in a (k, D)-refutation.

▶ **Definition 18** (Canonized Refutation). *Let D be a coherent DP-core, and $k \in \mathbb{N}$, $F = (f_1, \ldots, f_m)$ be a sequence of relabelings, and $R \equiv (\mathfrak{b}_0, S_0)(\mathfrak{b}_1, S_1) \ldots (\mathfrak{b}_m, S_m)$ be an F-relabeled (k, D)-refutation. We say that R is canonized if for each $i \in \mathbb{N}$, the pair $(\mathfrak{b}_i, S_i)$ is canonical.*

Our main theorem (Theorem 19) states that if D is a coherent DP-core, then the existence of a canonized (k, D)-refutation is equivalent to the existence of a graph of treewidth at most k that does not belong to the property $\mathbb{G}(\mathsf{D})$ defined by D.

▶ **Theorem 19.** *Let D be a coherent DP-core and $k \in \mathbb{N}$. Then there is a canonized (k, D)-refutation if and only if some graph of treewidth at most k does not belong to $\mathbb{G}(\mathsf{D})$.*

Proof. Let G be a graph of treewidth at most k that does not belong to $\mathbb{G}(\mathsf{D})$. Then, by Theorem 6, there is a (k, D)-refutation $R = (\mathfrak{b}_0, S_0)(\mathfrak{b}_1, S_1) \ldots (\mathfrak{b}_m, S_m)$. Let $R' = (\mathfrak{b}'_0, S'_0)(\mathfrak{b}'_1, S'_1) \ldots (\mathfrak{b}'_m, S'_m)$ be the sequence of pairs where for each $i \in \{0, 1, ..., m\}$, $(\mathfrak{b}'_i, S'_i)$ is the canonical form of $(\mathfrak{b}_i, S_i)$. Then R' is a canonized DP-refutation.

For the converse, suppose there is a canonized (k, D)-refutation R. By Definition 18, R is a F-relabeled refutation for some relabeling sequence F. By Theorem 17, there is a k-instructive tree decomposition τ where $\mathcal{G}(\tau) \notin \mathbb{G}(\mathsf{D})$. ◀

6 Early Pruning

Our second main contribution is an early-pruning procedure that can be applied in the study of conjectures of the form $\mathbb{P}_1 \to \mathbb{P}_2$ where $\mathbb{P}_1$ is closed under subgraphs. This reduces the search space even more because it allows us to avoid the computation of states that do not contribute in the search for a counter-example.

A graph property $\mathbb{P}$ is said to be closed under sub-graphs if whenever a graph G belongs to $\mathbb{P}$, we have that every sub-graph of G also belongs to $\mathbb{P}$. If the graph property $\mathbb{G}(\mathsf{D})$ specified by a given coherent DP-core D is of the form $\mathbb{G}(\mathsf{D}_1) \to \mathbb{G}(\mathsf{D}_2)$ for coherent DP-cores D_1 and D_2, such that $\mathbb{G}(\mathsf{D}_1)$ is closed under subgraphs, then when running our inference algorithm to determine whether some graph of treewidth at most k is not contained in $\mathbb{G}(\mathsf{D})$ we may prune the search earlier. The following simple, but crucial observation is the basis of our specialized search.

▶ **Observation 20.** *Let $\mathbb{P}$ be a graph property closed under subgraphs. Let τ and τ' be k-instructive tree decompositions such that τ is a subterm of τ'. Then, if $\mathcal{G}(\tau) \notin \mathbb{P}$, then $\mathcal{G}(\tau') \notin \mathbb{P}$.*

Proof. Assume that $\mathcal{G}(\tau) \notin \mathbb{P}$. Since τ is a subterm of τ', we have that $\mathcal{G}(\tau)$ is isomorphic to a subgraph of $\mathcal{G}(\tau')$. Suppose $\mathcal{G}(\tau') \in \mathbb{P}$, then since $\mathbb{P}$ is closed under subgraphs, we have that $\mathcal{G}(\tau) \in \mathbb{P}$. This contradicts the assumption that $\mathcal{G}(\tau) \notin \mathbb{P}$. ◀

Observation 20 implies that in order to determine whether there is a graph of treewidth at most k that does not belong to $\mathbb{G}(\mathsf{D})$, instead of searching for inconsistent (k, D)-pairs, we may instead search for inconsistent $(k, \mathsf{D}_1, \mathsf{D}_2)$-*triples*. Such a triple, is a triple of the form $(\mathfrak{b}, S_1, S_1)$ satisfying the following properties:

1. $(\mathfrak{b}, S_1)$ is a (k, D_1)-pair,
2. $(\mathfrak{b}, S_2)$ is a (k, D_2)-pair,
3. S_1 has a final local witness for D_1 but S_2 does not have a final local witness for D_2.

This allows a more efficient search because, since $\mathbb{G}(\mathsf{D}_1)$ is assumed to be closed under subgraphs, as soon as we have reached a $(k, \mathsf{D}_1, \mathsf{D}_2)$-triple $(\mathfrak{b}, S_1, S_2)$ where $(\mathfrak{b}, S_1)$ is an inconsistent (k, D_1)-pair, we know that no triple $(\mathfrak{b}, S_1', S_2')$ derived from $(\mathfrak{b}, S_1, S_2)$ will be inconsistent (because S_1' does not contain a final witness for D_1). Therefore, we do not need to consider $(k, \mathsf{D}_1, \mathsf{D}_2)$-triples derived from $(\mathfrak{b}, S_1, S_2)$. This construction is carried out in details in the full version of this work.

7 Reed's Conjecture Parameterized by Treewidth

In this section, we provide a concrete example of how dynamic programming algorithms can be used to provide asymptotic upper bounds on the time-complexity of verifying whether a given graph theoretic conjecture is valid on the class of graphs of width at most k. More specifically, we analyze the following well known conjecture due to Reed [21], which establishes an upper bound on the chromatic number $\chi(G)$ of a triangle-free graph G in terms of the maximum degree $\Delta(G)$ of G.

▶ **Conjecture 21.** *For any simple, triangle-free, undirected graph G, $\chi(G) \leq \lceil \frac{\Delta(G)+3}{2} \rceil$.*

It is worth noting that graphs of treewidth at most k are $(k+1)$-colorable [3]. The following theorem due to Dvorák and Kawarabayashi establishes a better upper bound for the chromatic number of triangle-free graphs in terms of treewidth.

▶ **Theorem 22** ([10])**.** *For any triangle-free graph G of treewidth $\leq k$, $\chi(G) \leq \lceil \frac{k+3}{2} \rceil$.*

Therefore, in order to prove that every graph of treewidth at most k satisfies Conjecture 21, it is enough to show that for each $s \in \{0, \ldots, k-1\}$, every graph of treewidth at most k and maximum degree at most s has chromatic number at most $\lceil \frac{s+3}{2} \rceil$, since for larger values of s, Theorem 22 implies that the conjecture is true. Now, let $\textsc{Colorable}_r$ denote the graph property consisting of all graphs that are r-colorable, $\textsc{MaxDeg}_d$ denote the graph property consisting of all graphs that have maximum degree at least d, $\textsc{Clique}_\omega$ be the property consisting of all graphs that have clique number at least ω, and $\textsc{MultiEdge}$ be the property consisting of all graphs that have some multiple edges. Let $\textsc{Reed}(s)$ be the graph property

$$\textsc{Reed}(s) \equiv \neg\textsc{MultiEdge} \wedge \neg\textsc{Clique}_3 \wedge \neg\textsc{MaxDeg}_{s+1} \rightarrow \textsc{Colorable}_{\lceil (s+3)/2 \rceil}. \qquad (2)$$

Then determining whether all graphs of treewidth at most k satisfy Reed conjecture is equivalent to determining whether for each $s \in \{0, 1, \ldots, k-1\}$, the set of all graphs of treewidth at most k is contained in property $\textsc{Reed}(s)$. Now, $\textsc{Reed}(s)$ is a Boolean combination of four properties, each of which can be decided coherent DP-cores whose bitlength is polynomial in k. The *bitlength of a DP-core* D is a function $\beta(k)$ that measures the maximum number of bits of a local witness produced during the processing of a k-instructive tree decomposition. The specification of each of the four DP-cores can be found in the appendix. Such implementations are of independent interest, because each such core may be viewed as an algorithm that takes a k-instructive tree decomposition τ as input and decides whether the graph $\mathcal{G}(\tau)$ associated with τ satisfies the corresponding property represented by the core.

▶ **Theorem 23** (DP-Cores for Reed's Conjecture). *There exist coherent, instructive DP-cores C-MaxDeg, C-Colorable, C-Clique$_\omega$, C-MultiEdge satisfying the following properties.*

1. *C-MaxDeg$_d$ has bit-length $O(k \cdot \log d)$ and $\mathbb{G}(C\text{-MaxDeg}_d) = \text{MaxDeg}_d$.*
2. *C-Colorable$_r$ has bit-length $O(k \cdot \log r)$ and $\mathbb{G}(C\text{-Colorable}_r) = \text{Colorable}_r$.*
3. *C-Clique$_\omega$ has bit-length $O(k \cdot \log \omega)$ and $\mathbb{G}(C\text{-Clique}_\omega) = \text{Clique}_\omega$.*
4. *C-MultiEdge has bit-length $O(k^2)$ and $\mathbb{G}(C\text{-MultiEdge}) = \text{MultiEdge}$*

Therefore, as a consequence of Theorem 19, Theorem 23, and Equation 2 we have the following corollary.

▶ **Corollary 24.** *For each $k \in \mathbb{N}$, Reed's conjecture for triangle-free graphs can be tested in time double-exponential in $O(k^2)$.*

Actually, the bound in Corollary 24 can be improved to double-exponential in $O(k \cdot \log k)$ by noting that the DP-core C-MultiEdge is deterministic.

8 Experimental Evaluation

We have implemented our width-based automated theorem proving framework on a software called *TreeWidzard*. Our software provides an interface that facilitates the implementation of dynamic programming algorithms parameterized by treewidth and pathwidth, and the integration of such algorithms with the purpose of width based automated theorem proving. In this section we evaluate our implementation on the task of producing counter-examples for wrong graph-theoretic statements, and also to provide a verification of Reed's conjectures on graphs of small treewidth and pathwidth.

8.1 Constructing Counterexamples for Wrong Mathematical Statements

One of the most remarkable applications of the framework of width-based automated theorem proving lies in its ability to search for counter-examples width-wise. The advantage of this approach is that the width of a graph may be significantly smaller than its number of vertices.

As stated in Theorem 22, triangle-free graphs of treewidth at most k have chromatic number at most $\lceil \frac{k+3}{2} \rceil$. It can be shown by an analytic proof that for each $k \geq 1$, this bound is tight [10], in the sense that there are graphs of treewidth at most k that cannot be properly colored with $\lceil \frac{k+3}{2} \rceil - 1$ colors. In Figure 1:Left we depict a triangle-free graph with 14 vertices, 27 edges and chromatic number 4. Apart from the drawing, for which we took some artistic liberty, this graph was obtained as a counter-example for the following statement: *triangle-free graphs have chromatic number at most* 3. In particular, the graph

was found when restricting our search to graphs of pathwidth at most 4. Note that it follows from [10] that this statement is true for graphs of pathwidth at most 3, and our search terminates without counter-examples in this case.

In a similar vein, the graph depicted in Figure 1:Right is a triangle-free graph of maximum degree 4 and chromatic number 4. This graph was obtained as a counter example for the following statement which is a (false) strengthening of Conjecture 21: *triangle-free graphs of maximum degree at most* 4 *have chromatic number at most* 3. The graph was found when restricting our search to graphs of pathwidth at most 4. It is worth noting that it also follows from [10] that this statement is true for graphs of pathwidth at most 3. In this case, our algorithm concluded the search without a counter-example.

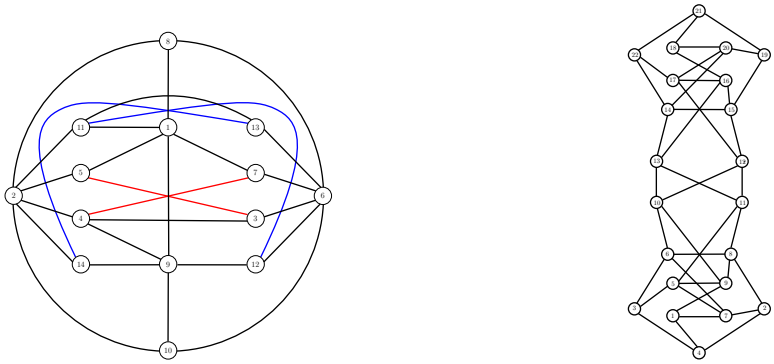


Figure 1 Left: G is a triangle-free graph of pathwidth 4 and chromatic number 4. This is a counter-example for a strengthening of a theorem of Reed and Dvorak. Right: G is a triangle-free graph of pathwidth 4, chromatic number 4, and maximum degree 4. This is a counter-example for a strengthening of Reed's conjecture for triangle free graphs.

8.2 Validating Reed's Triangle Free Conjecture on Graphs of Small Pathwidth and Treewidth

When testing whether a given graph-theoretic conjecture is valid on the class of graphs of pathwidth/treewidth at most k, our deduction algorithm may terminate without producing a counter-example. This means that the conjecture is valid on the class of graphs of pathwidth/treewidth at most k.

When combining both the symmetry-breaking technique introduced in Section 5 with the early-pruning technique introduced in Section 6, our software was able to confirm Conjecture 21 in the class of graphs of pathwidth at most 5. To the best of our knowledge, an analytic proof of Conjecture 21 on this class of graphs is lacking in the literature. It is worth noting that to confirm this case, it is enough to consider graphs of maximum degree at most 3, since for graphs of larger degree Reed's conjecture follow from Theorem 22.

In the verification of Conjecture 21 in the case of graphs of pathwidth at most 5, our search, using both techniques introduced in this work, produced 746187 states, took about 21 hours, and consumed 35 GB of memory on a cluster with processors of type Intel Xeon Gold 6130, with 2.1 Ghz. The search in this particular case was executed with 64 cores and 128 threads. It is worth noting that this case becomes prohibitively large if we deactivate either the canonization procedure or the subgraph-closed premise search. Therefore, the techniques introduced in this work were crucial in this regard.

For graphs of treewidth at most 3, and graphs of pathwidth at most 4, we performed a comparison between the original deduction algorithm introduced in [9], and the algorithm augmented only with the symmetry breaking procedure, only with the early pruning procedure,

and with both (See Table 1). Both improvements when applied isolated decrease significantly the number of states considered during the search process. Nevertheless, when combining both approaches, the reduction of the search space was very expressive. For example, in the case of pathwidth 3 and maximum degree 2, the method with no improvement produced more than 20 million states. The method with only the symmetry breaking improvement produced about 1 million states, the method with only the early pruning improvement produced about 1916 states, and the method with both improvements produced only 141 states.

Table 1 Number of states generated by our program when testing Reed's (ω, Δ, χ)-conjecture for $\omega = 2$ (triangle-free case) on graphs of constant pathwidth and treewidth. The entries in blue correspond to experiments that did not terminate, and the entries with a star correspond to experiments that stopped due to a memory limit.

	BFS			BFS-premise		
pw \ Δ	2	3	4	2	3	4
pw = 2	2503	4814	9877	149	341	617
pw = 3	> 20738085*	> 21164080*	5193467	1916	9850	27720
pw = 4	> 9463445*	> 6019042*	> 8333258*	29184	1156954	2438694
	ISO-BFS			ISO-BFS-premise		
pw = 2	520	945	1843	38	76	128
pw = 3	1031545	1050960	223920	141	579	1451
pw = 4	> 9449990*	> 9903864*	> 5029614*	486	12375	24494
	ISO-BFS			ISO-BFS-premise		
tw = 2	>25351	>25960	>27632	57	153	330
tw = 3	>69621	>71183	>72891	194	1268	4080
tw = 4	>71858	>73532	>75426	616	>4942	>13438

9 Conclusion

In this work, we have given the first steps towards evaluating the width-based automated theorem proving approach introduced in [9] from a practical perspective. At the same time, we have introduced two techniques that have together drastically reduced the space of states to be explored during the search for a counter-example. While the first technique is quite general and can be applied in the study of any conjecture involving DP-cores for which a witness action can be defined, the second improvement can be applied in the case of conjectures of the form $\mathbb{P}_1 \to \mathbb{P}_2$, where $\mathbb{P}_1$ is a property closed under subgraphs. To illustrate the applicability of our methods, we have used our implementation to produce counter-examples for false graph-theoretic statements, and also to confirm Reed's triangle-free conjecture on the class of graphs of pathwidth at most 5 and on the class of graphs of treewidth at most 3.

It is worth highlighting the modularity of our approach. While the implementation of instructive dynamic programming cores requires specialized knowledge from the part of the programmer, the use of such cores once they have been implemented is straightforward. For instance, in our framework, the critical case for testing Reed's conjecture on graphs of pathwidth at most 5 is the case where the degree is at most 3. More precisely, a conjecture stating that all simple, triangle-free graphs of maximum degree at most 3 have chromatic number at most 3. This conjecture is stated in our framework using the following intuitive lines of code, where the first four lines correspond each to a graph property, and the last line corresponds to the conjecture being tested.

```
x := MaxDegree_AtLeast(4)
y := CliqueNumber_AtLeast(3)
z := HasMultipleEdges
w := Colorable(3)
Formula
NOT x AND NOT y AND NOT z IMPLIES w
```

The idea is that dynamic programming cores deciding graph properties are implemented as plugins that need to be implemented only once by a specialist and then used without difficulty by graph theorists. We believe that our approach has the potential to create a nice interchange of knowledge between the community of researchers working on parameterized complexity theory, and researchers working in automated theorem proving. In essence, our framework shows that the 3 decades of accumulated knowledge obtained in the development of faster width-based parameterized algorithms for model checking may now be put into use in the context of automated theorem proving.

References

1 Therese C. Biedl. On triangulating k-outerplanar graphs. *Discret. Appl. Math.*, 181(1):275–279, 2015. doi:10.1016/j.dam.2014.10.017.

2 Hans L. Bodlaender. *Classes of graphs with bounded tree-width*, volume 86. Unknown Publisher, 1986.

3 Hans L Bodlaender and Fedor V Fomin. Equitable colorings of bounded treewidth graphs. *Theoretical Computer Science*, 349(1):22–30, 2005.

4 Béla Bollobás, Paul A Catlin, and Paul Erdös. Hadwiger's conjecture is true for almost every graph. *Eur. J. Comb.*, 1(3):195–199, 1980.

5 Andreas Brandstädt, Van Bang Le, and Jeremy P Spinrad. *Graph classes: a survey*. SIAM, 1999.

6 CN Campos and Yoshiko Wakabayashi. On dominating sets of maximal outerplanar graphs. *Discrete Applied Mathematics*, 161(3):330–335, 2013.

7 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

8 Bruno Courcelle and Irène Durand. Automata for the verification of monadic second-order graph properties. *Journal of applied logic*, 10(4):368–409, 2012.

9 Mateus de Oliveira Oliveira and Farhad Vadiee. From width-based model checking to width-based automated theorem proving. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37(5), pages 6297–6304, 2023.

10 Zdeněk Dvořák and Ken-ichi Kawarabayashi. Triangle-free graphs of tree-width t are $\lceil (t+3)/2 \rceil$-colorable. *European Journal of Combinatorics*, 66:95–100, 2017.

11 DJ Guan and Xuding Zhu. Game chromatic number of outerplanar graphs. *Journal of Graph Theory*, 30(1):67–70, 1999.

12 Daniel Heldt, Kolja Knauer, and Torsten Ueckerdt. On the bend-number of planar and outerplanar graphs. *Discrete Applied Mathematics*, 179:109–119, 2014.

13 Pavol Hell and Xuding Zhu. The circular chromatic number of series-parallel graphs. *Journal of Graph Theory*, 33(1):14–24, 2000.

14 Jesper Lykke Jacobsen and Jesús Salas. Is the five-flow conjecture almost false? *Journal of Combinatorial Theory, Series B*, 103(4):532–565, 2013.

15 Frank Kammer. Determining the smallest k such that g is k-outerplanar. In *European Symposium on Algorithms*, pages 359–370. Springer, 2007.

16 Ton Kloks. *Treewidth: computations and approximations*. Springer, 1994.

17 Zhishi Pan and Xuding Zhu. Tight relation between the circular chromatic number and the girth of series-parallel graphs. *Discrete mathematics*, 254(1-3):393–404, 2002.

18 Léo Vieira Peres and Ricardo Dahab. Tutte's 3-flow conjecture for almost even graphs. *Procedia Computer Science*, 195:280–288, 2021.

19 Alexandre Pinlou and Éric Sopena. Oriented vertex and arc colorings of outerplanar graphs. *Information Processing Letters*, 100(3):97–104, 2006.

20 Andrzej Proskurowski and Jan Arne Telle. From bandwidth to pathwidth k. *_THEORETICAL-E*, page 90, 1996.

21 Bruce Reed. ω, δ, and χ. *Journal of Graph Theory*, 27(4):177–212, 1998.

22 Bo Zhou. Upper bounds for the zagreb indices and the spectral radius of series-parallel graphs. *International Journal of Quantum Chemistry*, 107(4):875–878, 2007.